SESSION 4

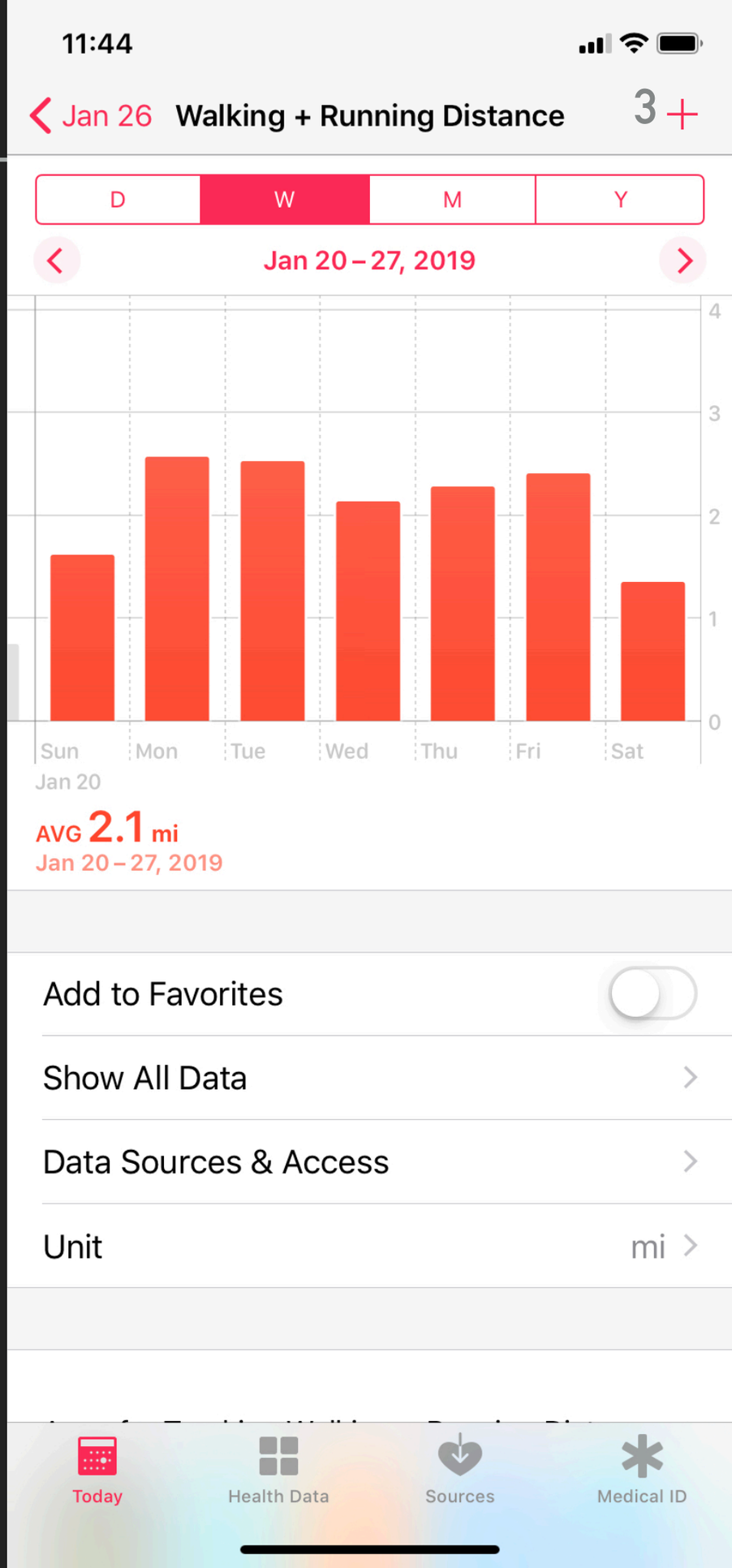
iOS DEVELOPMENT

BEAUTIFUL VIEWS WITH

CUSTOM DRAWING

WHY CUSTOM DRAWING?

- What are some ways you could create the bar graph?
- Consider:
  - Orange bars
  - Grid background

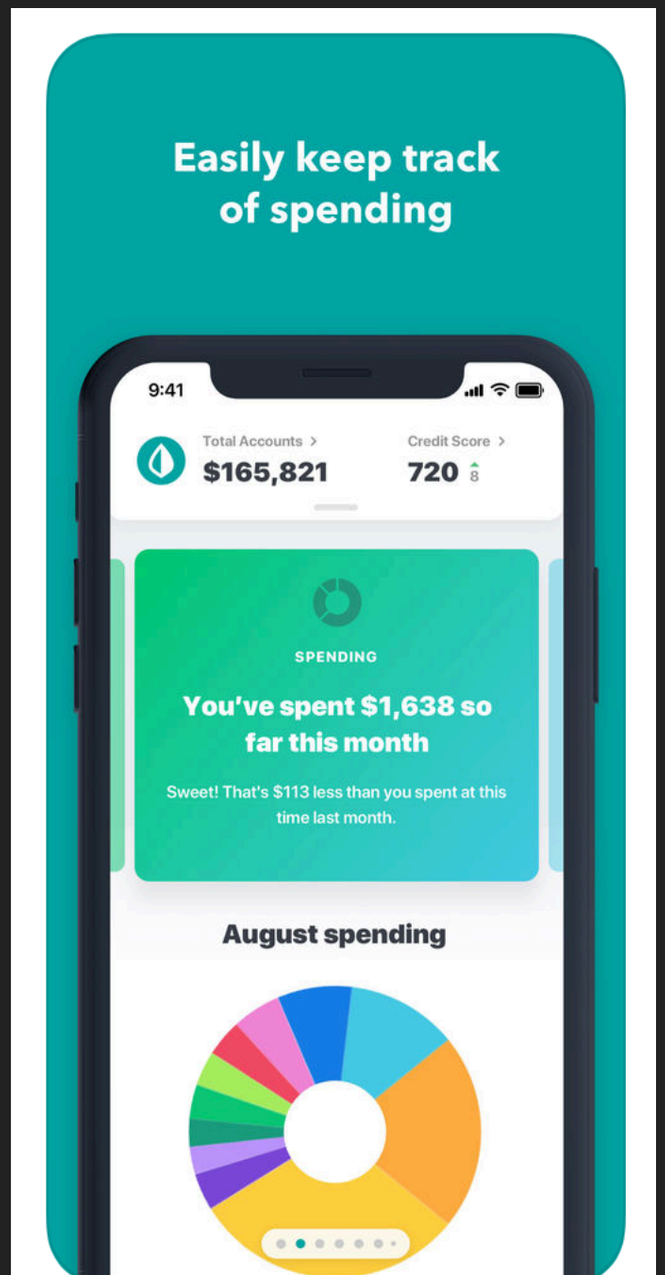


WHY CUSTOM DRAWING?

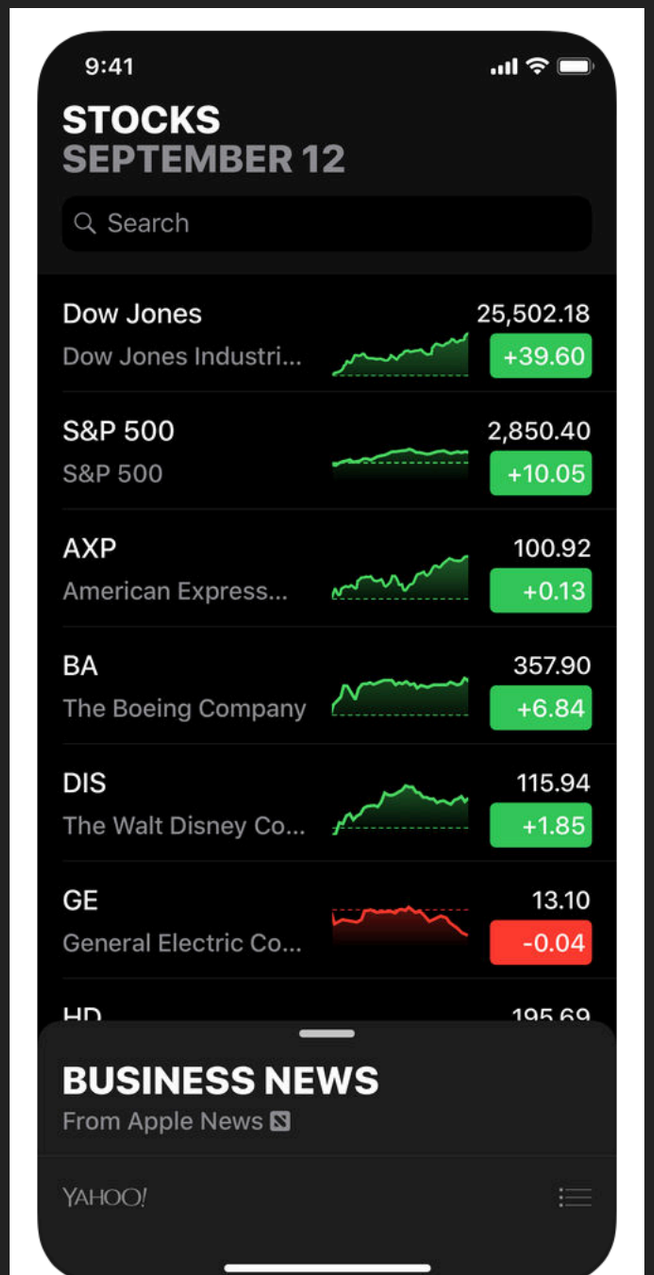
- What's more complex about this one?

EXAMPLES OF CUSTOM DRAWING

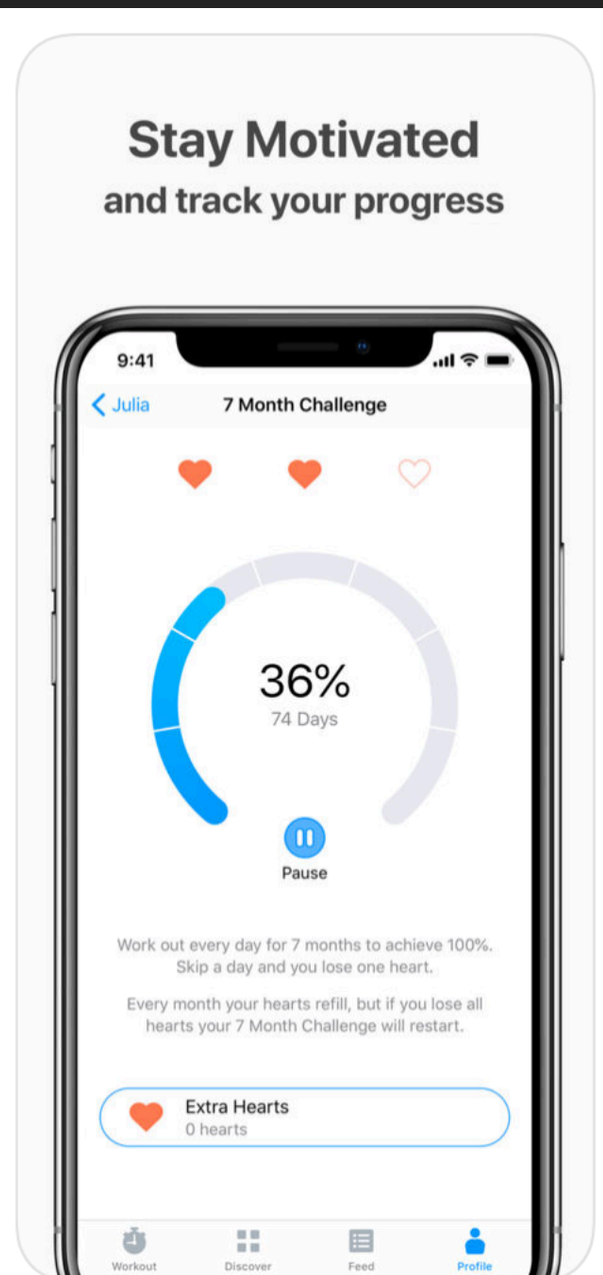
Mint



Stocks



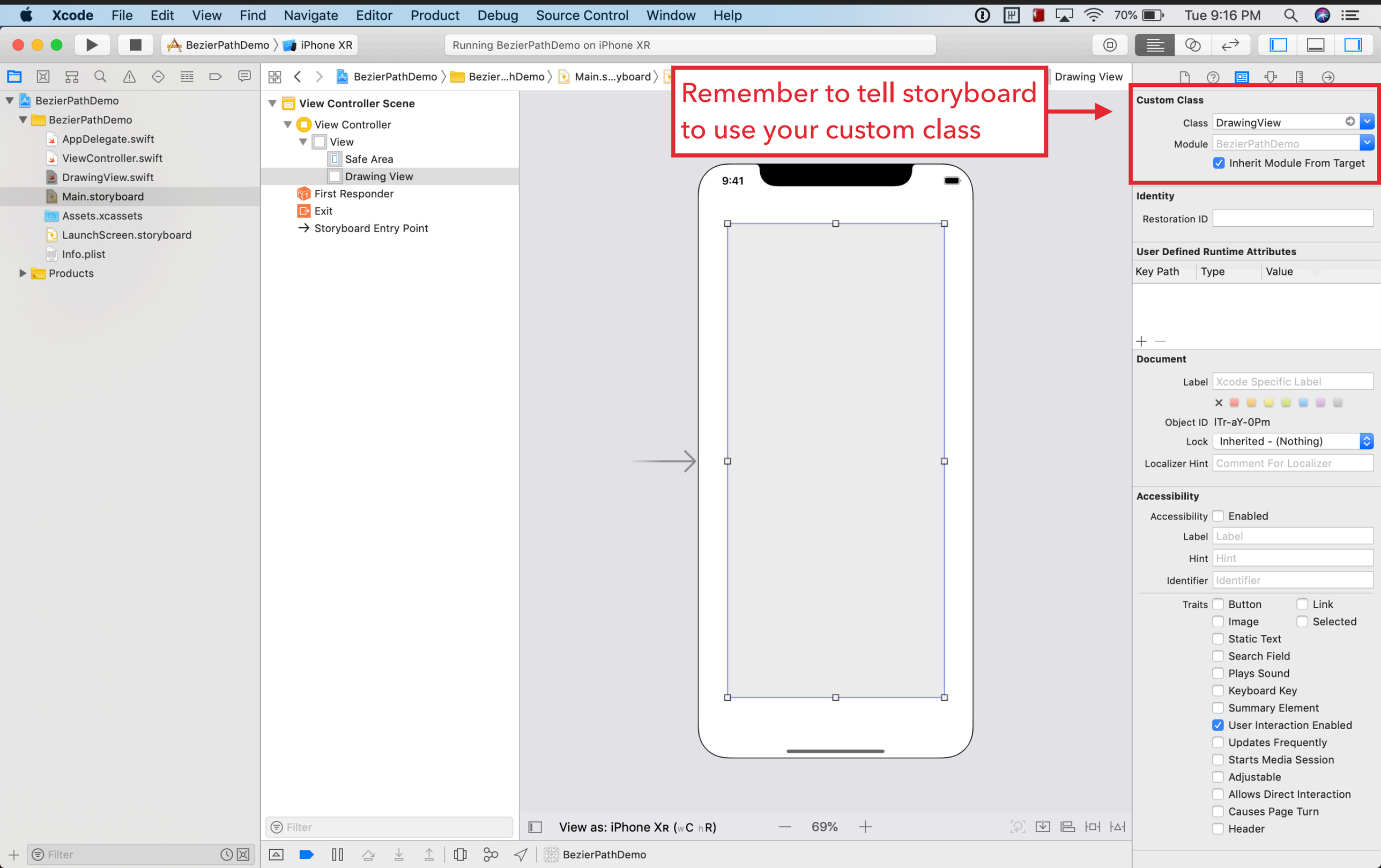
Seven



GETTING STARTED

- Custom drawing always takes place inside a `UIView`
- You'll need to:
  - Create a subclass of `UIView`
  - Override the `draw` method

UIBezierPath



Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

Session4Demos 〉 iPhone 11 Pro Max | Finished running Session4Demos on iPhone 11 Pro Max | 5

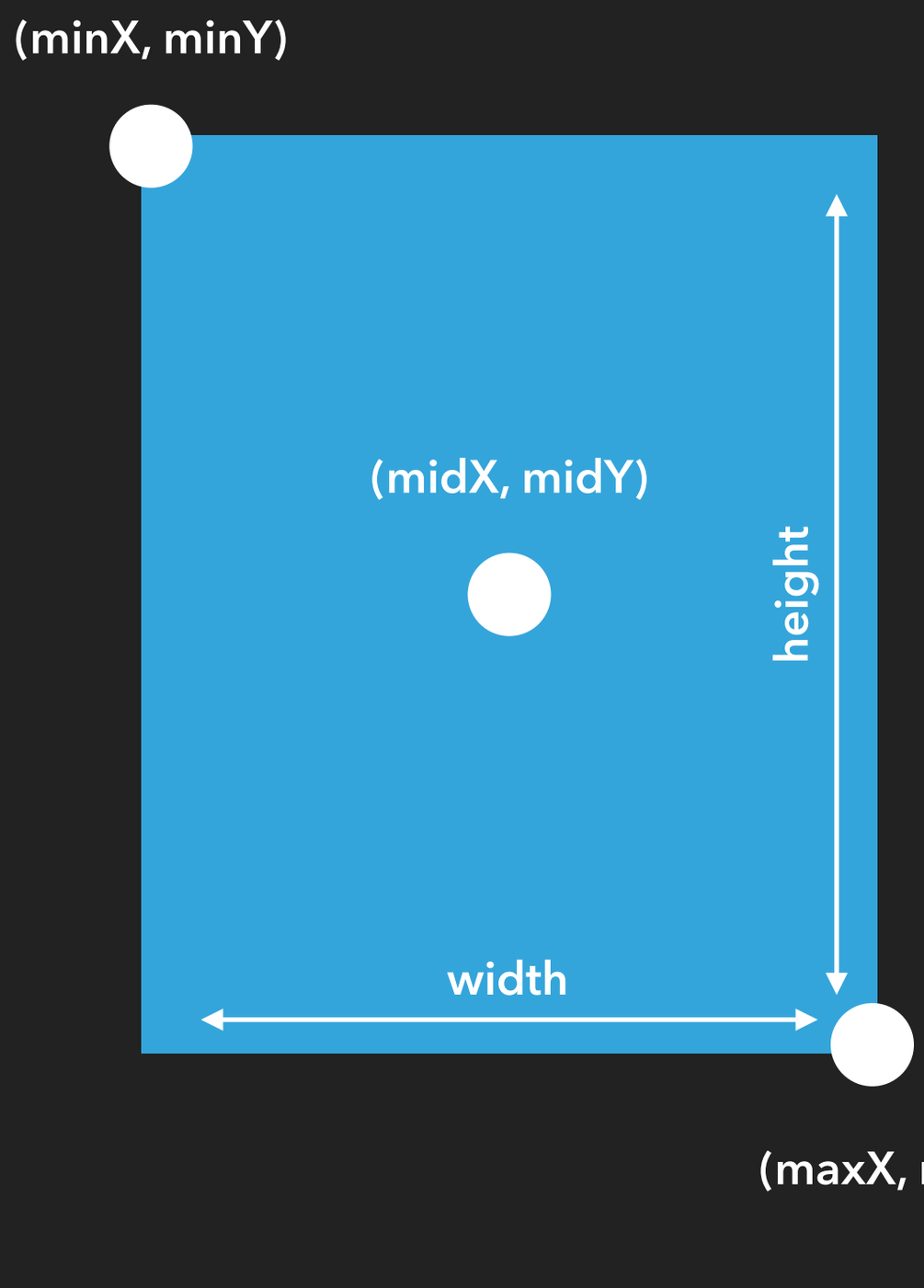
Session4Demos 〉 Session4Demos 〉 Drawing 〉 DrawingView.swift 〉 No Selection

- Session4Demos
  - Session4Demos
    - Drawing
      - DrawingView.swift
    - Layers
    - Animations 1
    - Gestures Recognizers
    - Animations 2
    - AppDelegate.swift
    - SceneDelegate.swift
    - Main.storyboard
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
  - Products

```swift
//
//  DrawingView.swift
//  Session4Demos
//
//  Created by Susan Stevens on 1/25/20.
//  Copyright © 2020 Susan Stevens. All rights reserved.
//

import UIKit

class DrawingView: UIView {

    override func draw(_ rect: CGRect) {
        // custom drawing code
    }
}
```

`rect` defines the dimensions of your canvas

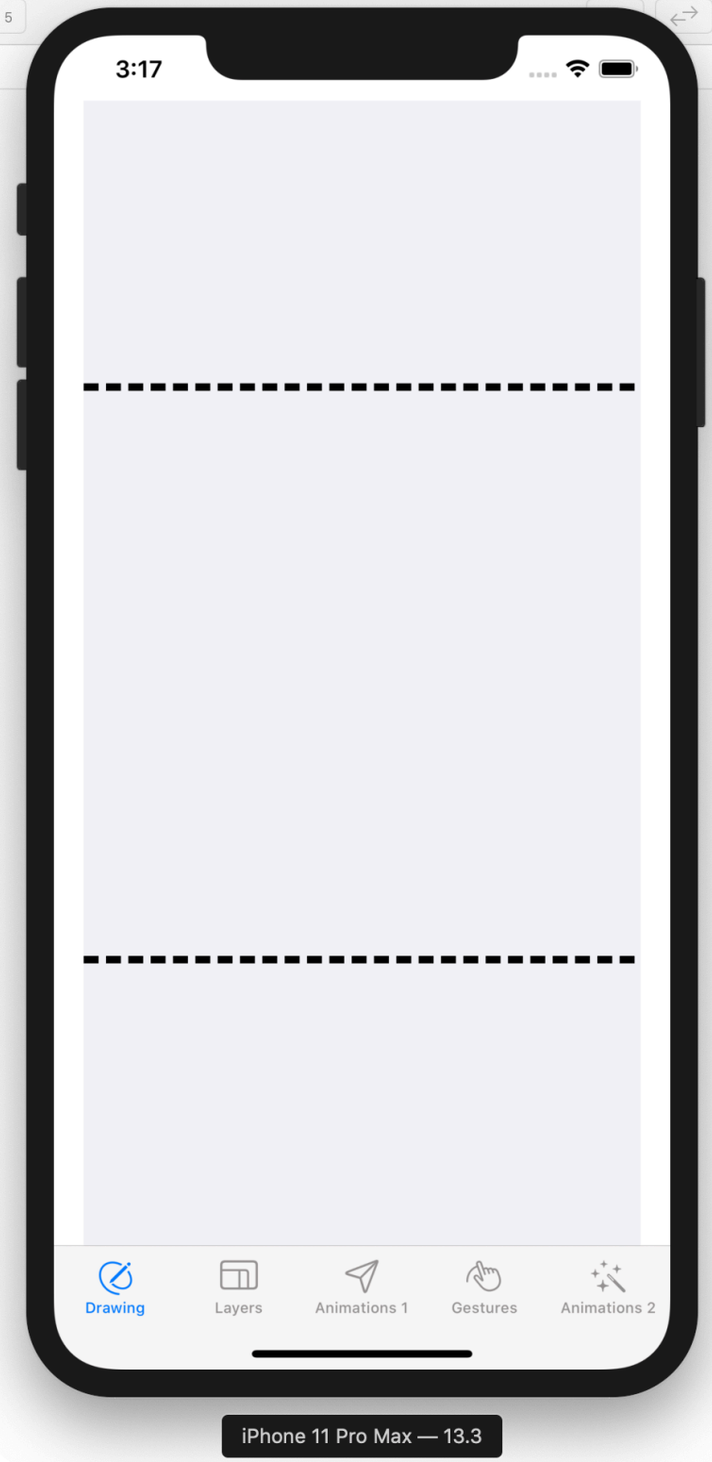
PROPERTIES OF CGRECT



UIBezierPath


```
//
//  Created by Susan Stevens on 1/25/20.
//  Copyright © 2020 Susan Stevens. All rights reserved.
//

import UIKit

class DrawingView: UIView {

    override func draw(_ rect: CGRect) {
        let width = rect.width
        let height = rect.height

        // Draw dashed horizontal lines
        let lines = UIBezierPath()

        lines.move(to: CGPoint(x: 0, y: height / 4))
        lines.addLine(to: CGPoint(x: width, y: height / 4))

        lines.move(to: CGPoint(x: 0, y: 3 * height / 4))
        lines.addLine(to: CGPoint(x: width, y: 3 * height / 4))

        lines.setLineDash([10, 5], count: 2, phase: 0)
        lines.lineWidth = 5
        lines.stroke()
```

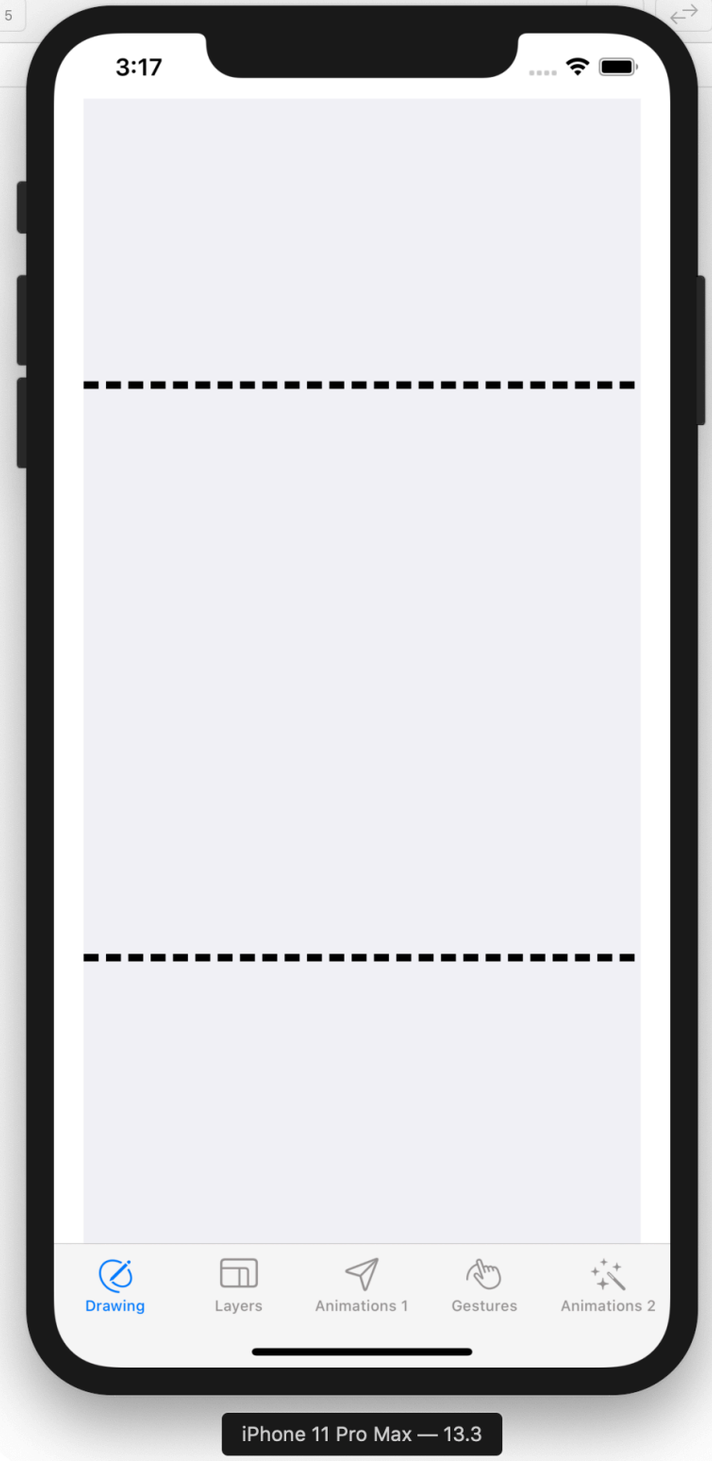
Create bezier path



UIBezierPath


```
//
//  Created by Susan Stevens on 1/25/20.
//  Copyright © 2020 Susan Stevens. All rights reserved.
//

import UIKit

class DrawingView: UIView {

    override func draw(_ rect: CGRect) {
        let width = rect.width
        let height = rect.height

        // Draw dashed horizontal lines
        let lines = UIBezierPath()

        lines.move(to: CGPoint(x: 0, y: height / 4))
        lines.addLine(to: CGPoint(x: width, y: height / 4))

        lines.move(to: CGPoint(x: 0, y: 3 * height / 4))
        lines.addLine(to: CGPoint(x: width, y: 3 * height / 4))

        lines.setLineDash([10, 5], count: 2, phase: 0)
        lines.lineWidth = 5
        lines.stroke()
```

Draw the top line



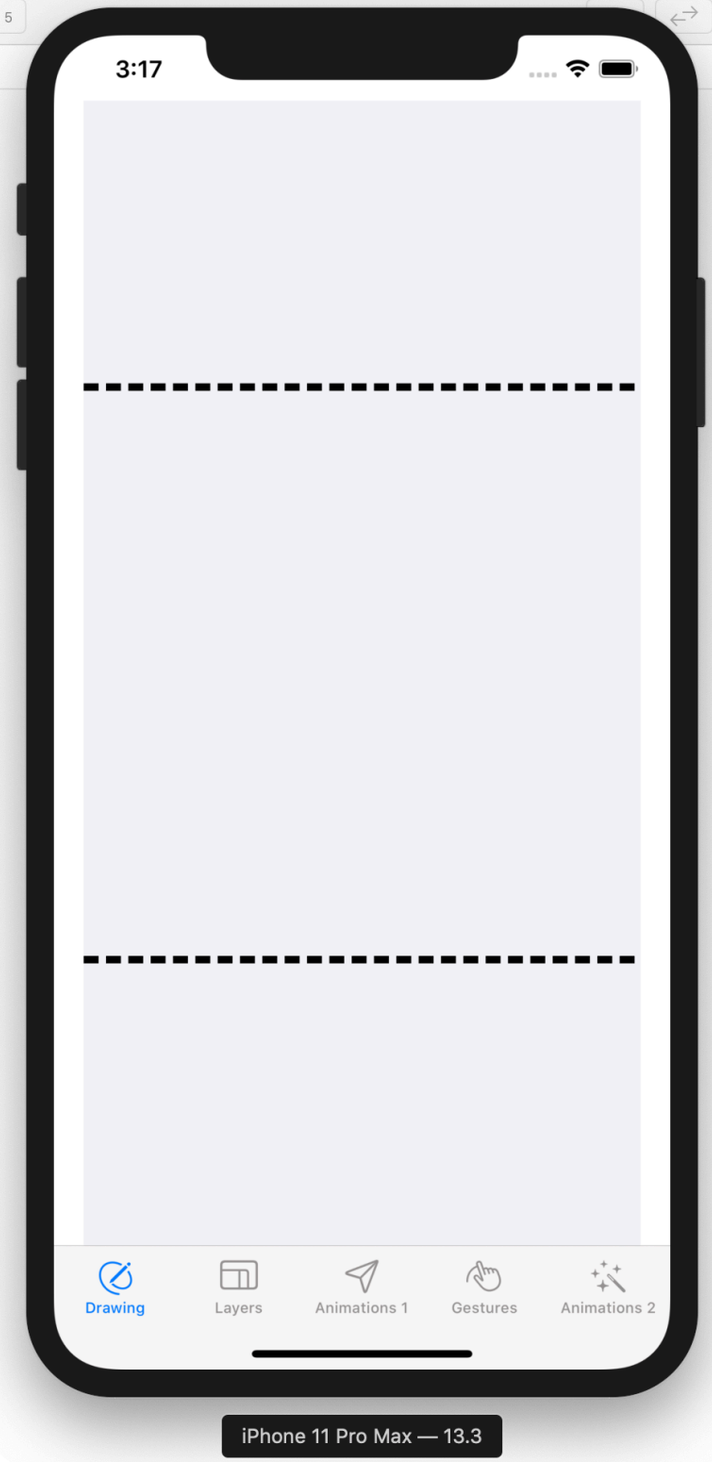
MOVE VS. ADD LINE

- `move` means “put the pencil down at this point”
- `addLine` means “draw a line to this point “

```
path.move(to: CGPoint(x: width / 2, y: 0))
path.addLine(to: CGPoint(x: width / 2, y: height))
```

```
//
//  Created by Susan Stevens on 1/25/20.
//  Copyright © 2020 Susan Stevens. All rights reserved.
//

import UIKit

class DrawingView: UIView {

    override func draw(_ rect: CGRect) {
        let width = rect.width
        let height = rect.height

        // Draw dashed horizontal lines
        let lines = UIBezierPath()

        lines.move(to: CGPoint(x: 0, y: height / 4))
        lines.addLine(to: CGPoint(x: width, y: height / 4))

        lines.move(to: CGPoint(x: 0, y: 3 * height / 4))
        lines.addLine(to: CGPoint(x: width, y: 3 * height / 4))

        lines.setLineDash([10, 5], count: 2, phase: 0)
        lines.lineWidth = 5
        lines.stroke()
```

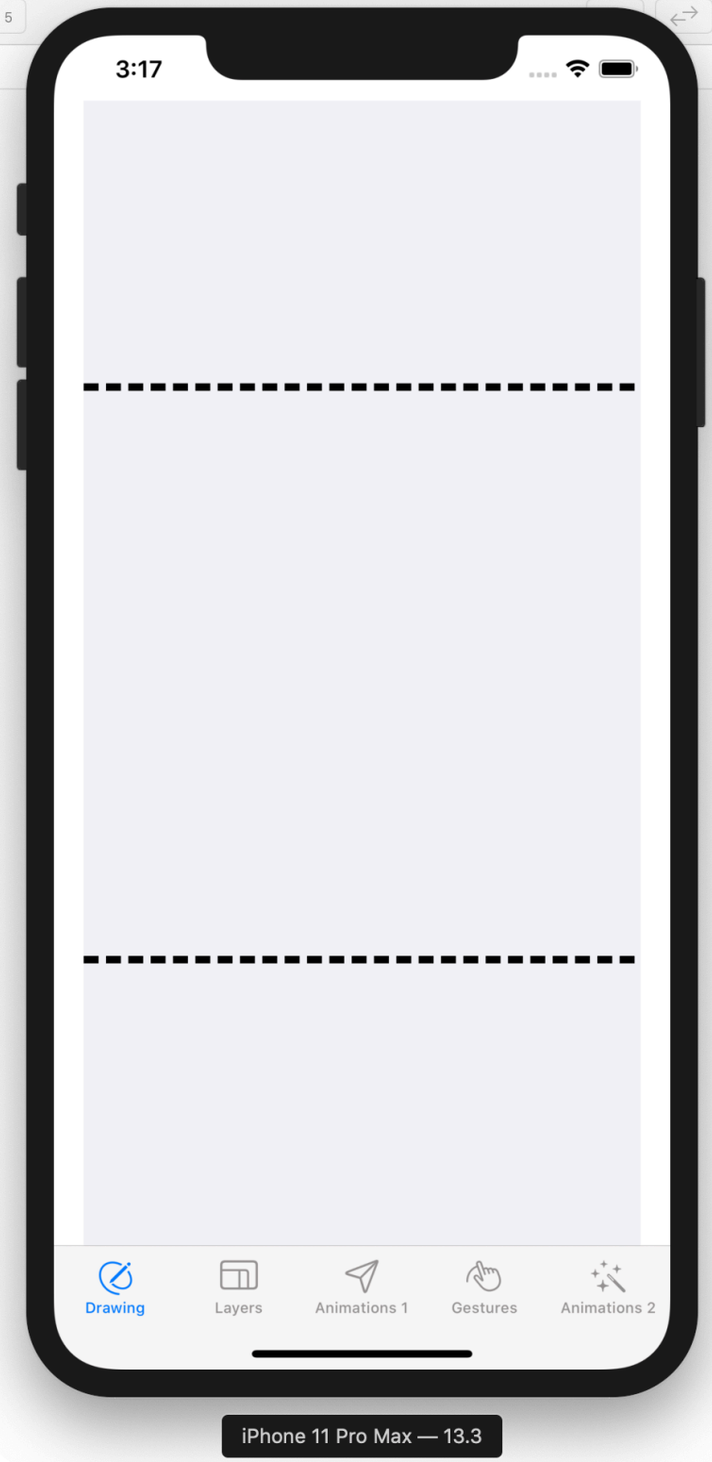
Draw the bottom line



UIBezierPath


```
//
//  Created by Susan Stevens on 1/25/20.
//  Copyright © 2020 Susan Stevens. All rights reserved.
//

import UIKit

class DrawingView: UIView {

    override func draw(_ rect: CGRect) {
        let width = rect.width
        let height = rect.height

        // Draw dashed horizontal lines
        let lines = UIBezierPath()

        lines.move(to: CGPoint(x: 0, y: height / 4))
        lines.addLine(to: CGPoint(x: width, y: height / 4))

        lines.move(to: CGPoint(x: 0, y: 3 * height / 4))
        lines.addLine(to: CGPoint(x: width, y: 3 * height / 4))

        lines.setLineDash([10, 5], count: 2, phase: 0)
        lines.lineWidth = 5
        lines.stroke()
```

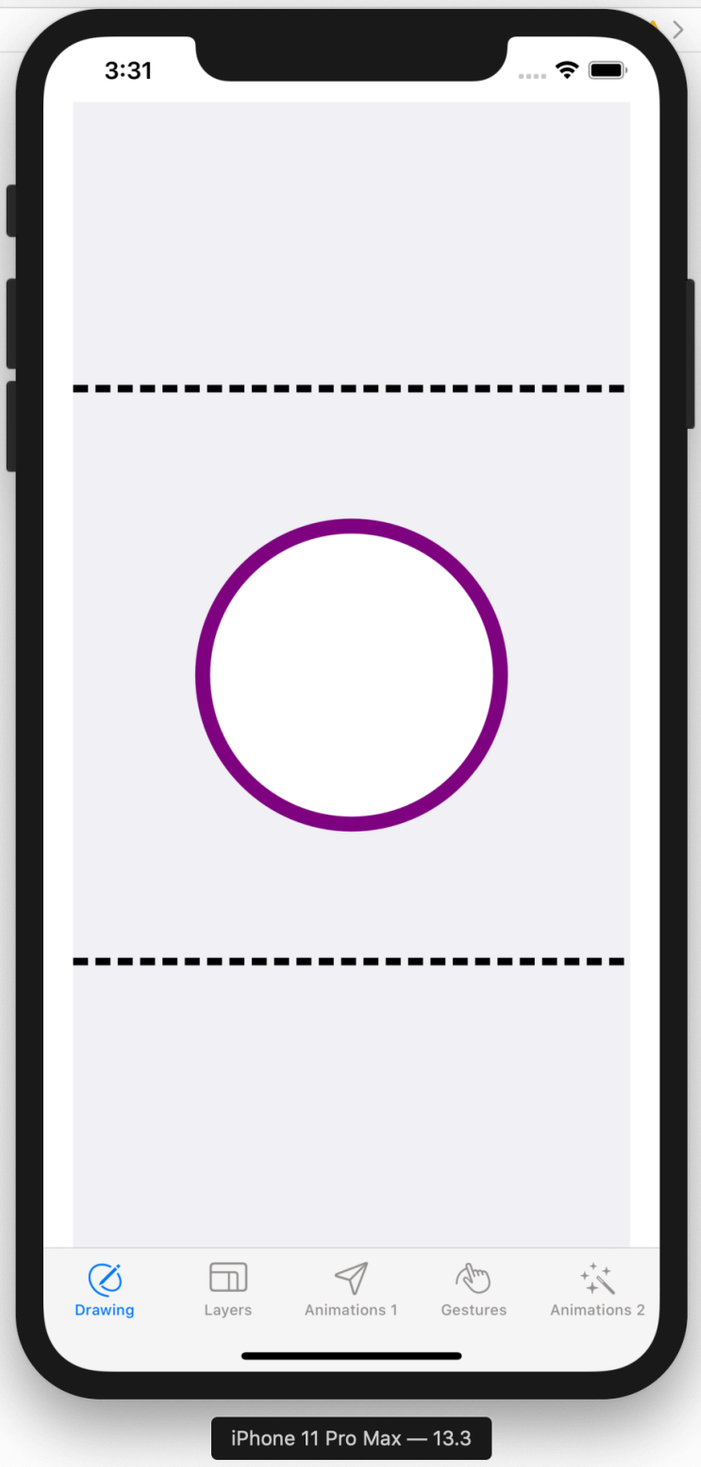
Style and draw the line



UIBezierPath

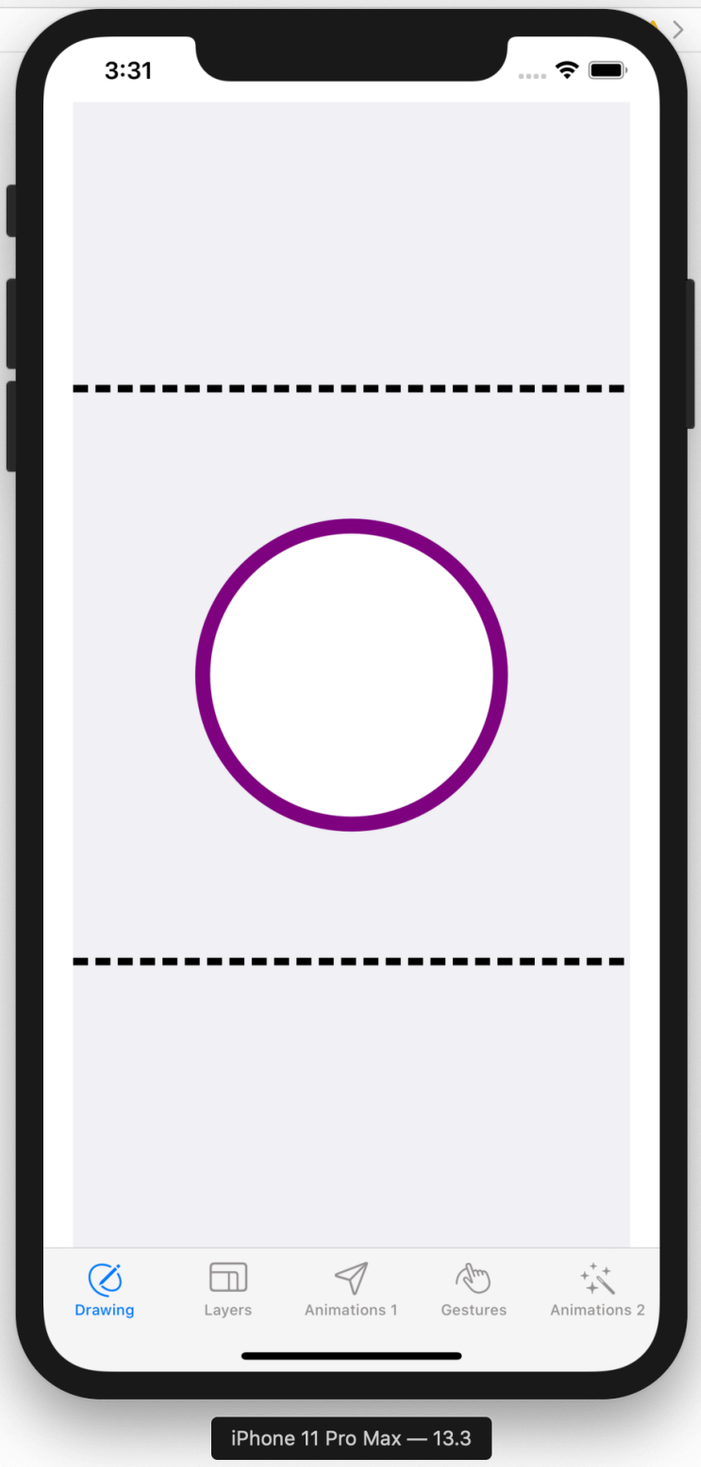

```
16
17          // Draw dashed horizontal lines
18          let lines = UIBezierPath()
19
20          lines.move(to: CGPoint(x: 0, y: height / 4))
21          lines.addLine(to: CGPoint(x: width, y: height / 4))
22
23          lines.move(to: CGPoint(x: 0, y: 3 * height / 4))
24          lines.addLine(to: CGPoint(x: width, y: 3 * height / 4))
25
26          lines.setLineDash([10, 5], count: 2, phase: 0)
27          lines.lineWidth = 5
28          lines.stroke()
29
30          // Draw a circle
31          let circle = UIBezierPath(ovalIn: CGRect(x: width / 2 - 100,
32                                                   y: height / 2 - 100,
33                                                   width: 200,
34                                                   height: 200))
35
36          UIColor.white.setFill()
37          circle.fill()
38
39          UIColor.purple.setStroke()
40          circle.lineWidth = 10
41          circle.stroke()
42
43
44
```

Create a circular path



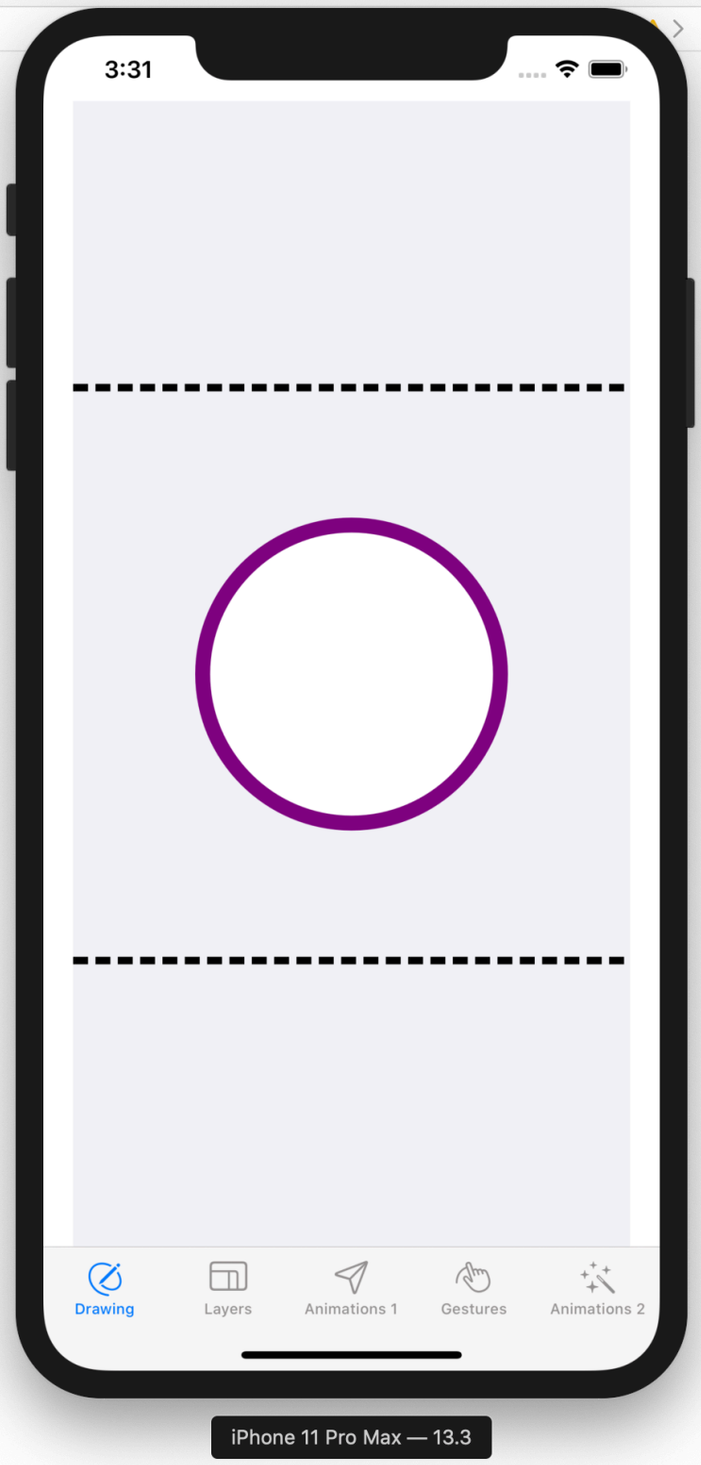
```
16
17          // Draw dashed horizontal lines
18          let lines = UIBezierPath()
19
20          lines.move(to: CGPoint(x: 0, y: height / 4))
21          lines.addLine(to: CGPoint(x: width, y: height / 4))
22
23          lines.move(to: CGPoint(x: 0, y: 3 * height / 4))
24          lines.addLine(to: CGPoint(x: width, y: 3 * height / 4))
25
26          lines.setLineDash([10, 5], count: 2, phase: 0)
27          lines.lineWidth = 5
28          lines.stroke()
29
30          // Draw a circle
31          let circle = UIBezierPath(ovalIn: CGRect(x: width / 2 - 100,
32                                                   y: height / 2 - 100,
33                                                   width: 200,
34                                                   height: 200))
35
36          UIColor.white.setFill()
37          circle.fill()
38
39          UIColor.purple.setStroke()
40          circle.lineWidth = 10
41          circle.stroke()
42
43
44
```

Fill with white



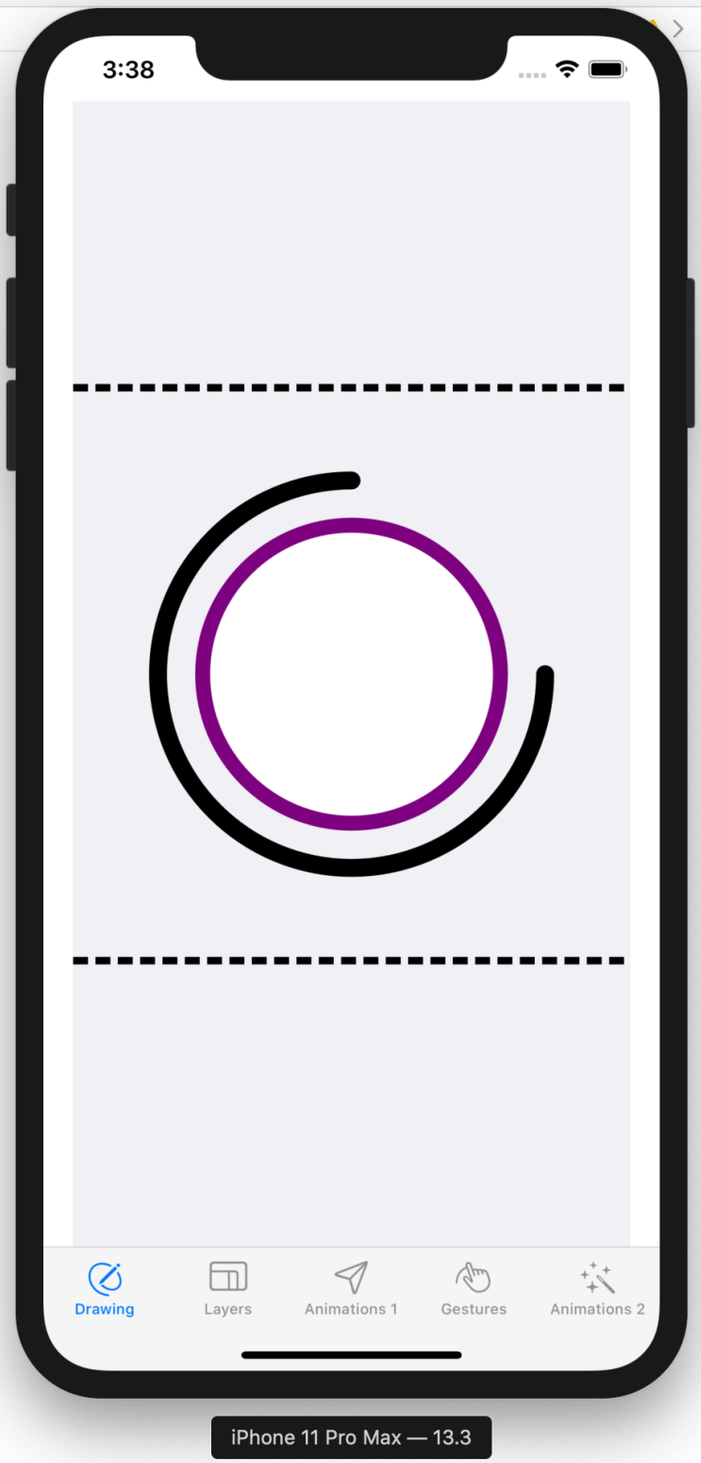
```
16
17          // Draw dashed horizontal lines
18          let lines = UIBezierPath()
19
20          lines.move(to: CGPoint(x: 0, y: height / 4))
21          lines.addLine(to: CGPoint(x: width, y: height / 4))
22
23          lines.move(to: CGPoint(x: 0, y: 3 * height / 4))
24          lines.addLine(to: CGPoint(x: width, y: 3 * height / 4))
25
26          lines.setLineDash([10, 5], count: 2, phase: 0)
27          lines.lineWidth = 5
28          lines.stroke()
29
30          // Draw a circle
31          let circle = UIBezierPath(ovalIn: CGRect(x: width / 2 - 100,
32                                                   y: height / 2 - 100,
33                                                   width: 200,
34                                                   height: 200))
35
36          UIColor.white.setFill()
37          circle.fill()
38
39          UIColor.purple.setStroke()
40          circle.lineWidth = 10
41          circle.stroke()
42
43
44
```

Add a purple border



```
        // Draw a circle
        let circle = UIBezierPath(ovalIn: CGRect(x: width / 2 - 100,
                                                 y: height / 2 - 100,
                                                 width: 200,
                                                 height: 200))

        UIColor.white.setFill()
        circle.fill()

        UIColor.purple.setStroke()
        circle.lineWidth = 10
        circle.stroke()

        // Draw an arc
        let arcCenter = CGPoint(x: width / 2, y: height / 2)
        let arc = UIBezierPath(arcCenter: arcCenter,
                               radius: 130,
                               startAngle: 0,
                               endAngle: CGFloat.pi * 3/2,
                               clockwise: true)

        UIColor.black.setStroke()
        arc.lineWidth = 12
        arc.lineCapStyle = .round
        arc.stroke()
    }
}
```

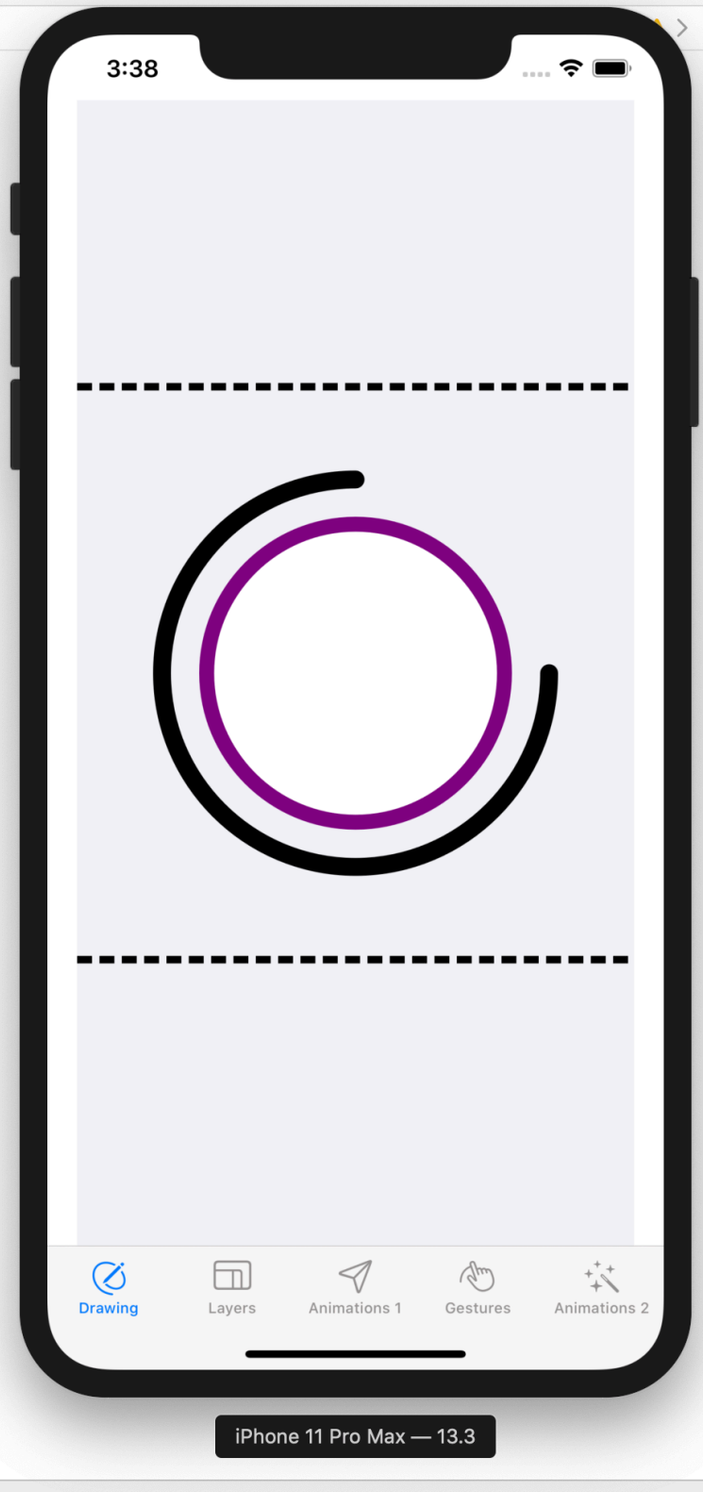
Create a curved line



UIBezierPath


```
        // Draw a circle
        let circle = UIBezierPath(ovalIn: CGRect(x: width / 2 - 100,
                                                 y: height / 2 - 100,
                                                 width: 200,
                                                 height: 200))

        UIColor.white.setFill()
        circle.fill()

        UIColor.purple.setStroke()
        circle.lineWidth = 10
        circle.stroke()

        // Draw an arc
        let arcCenter = CGPoint(x: width / 2, y: height / 2)
        let arc = UIBezierPath(arcCenter: arcCenter,
                               radius: 130,
                               startAngle: 0,
                               endAngle: CGFloat.pi * 3/2,
                               clockwise: true)

        UIColor.black.setStroke()
        arc.lineWidth = 12
        arc.lineCapStyle = .round
        arc.stroke()
    }
}
```

Style and draw the arc



REMINDER

- You must call `path.stroke()` or `path.fill()` for your bezier path to be rendered on screen

HOW DOES IT WORK?

- `UIBezierPath` is a wrapper around `CGPath`, which comes from Core Graphics.
- Core Graphics is a lower-level framework. UIKit abstracts away some of the complexity.
  - Note: The “CG” prefix on `CGRect` and `CGPoint` stands for Core Graphics

HOW DOES IT WORK?

- All drawing done by Core Graphics must take place inside a `CGContext`.

```
24  class CustomView: UIView {
25
26      func someMethod() {
27          // this won't work!
28          let path = UIBezierPath()
29
30          path.move(to: CGPoint(x: 100, y: 0))
31          path.addLine(to: CGPoint(x: 100, y: 50))
32
33          UIColor.green.setStroke()
34          path.stroke()
35      }
36
```

HOW DOES IT WORK?

- The `draw` method implicitly provides a context for you.

```
override func draw(_ rect: CGRect) {
    // create circle path
    let circlePath = UIBezierPath(ovalIn: frame)

    // set fill color
    UIColor.blue.setFill()

    // fill path
    circlePath.fill()

    let width = frame.size.width
    let height = frame.size.height

    // draw vertical line
    let path = UIBezierPath()
```

HOW DOES IT WORK?

- To trigger a re-draw of a view, do not call `draw` directly
  - Instead, call `myView.setNeedsDisplay()` if you need to tell your view to re-draw itself

BEAUTIFUL VIEWS WITH

LAYERS

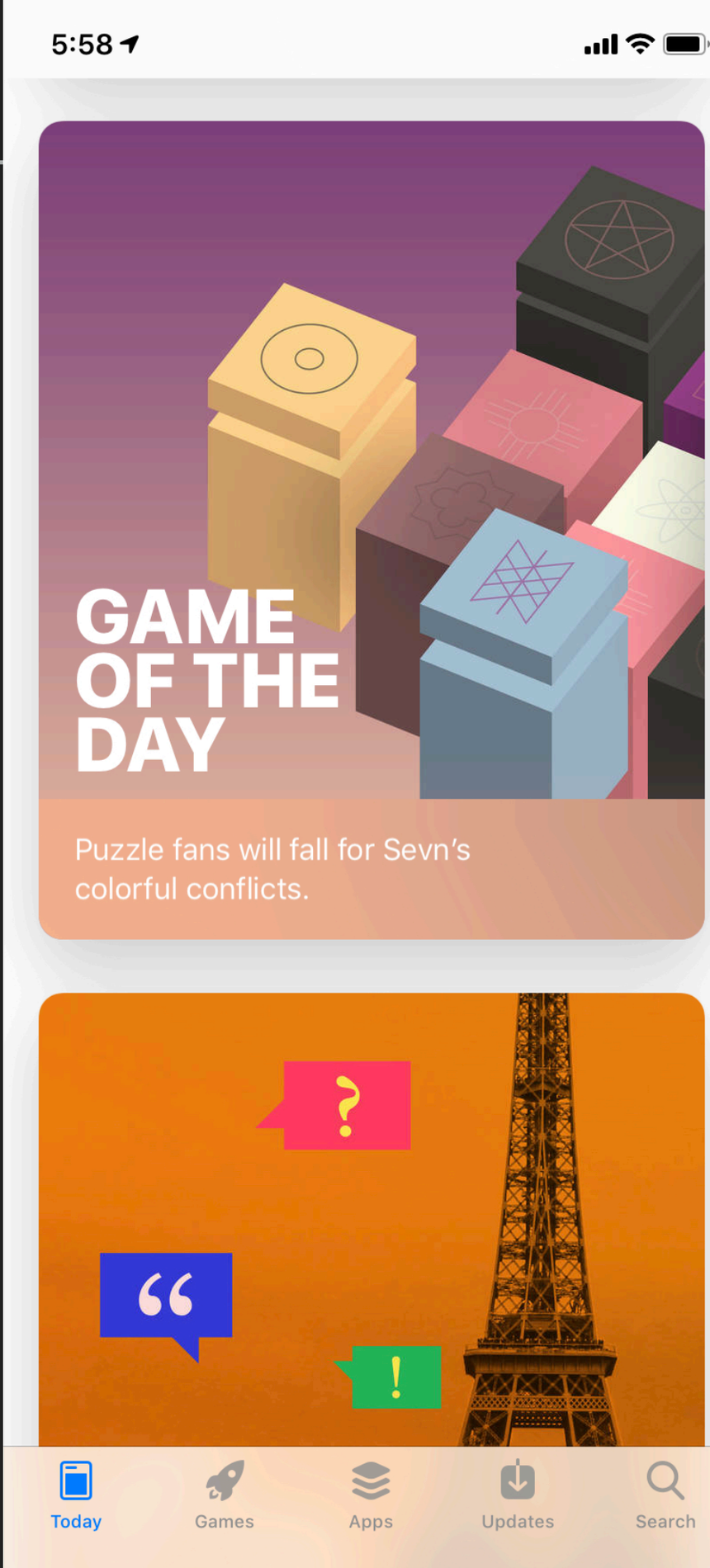
VISUAL PROPERTIES OF UIVIEW

- So far, we've talked about:
  - Frame & bounds
  - Background color
  - Alpha

WHAT ABOUT...

- Rounded corners?

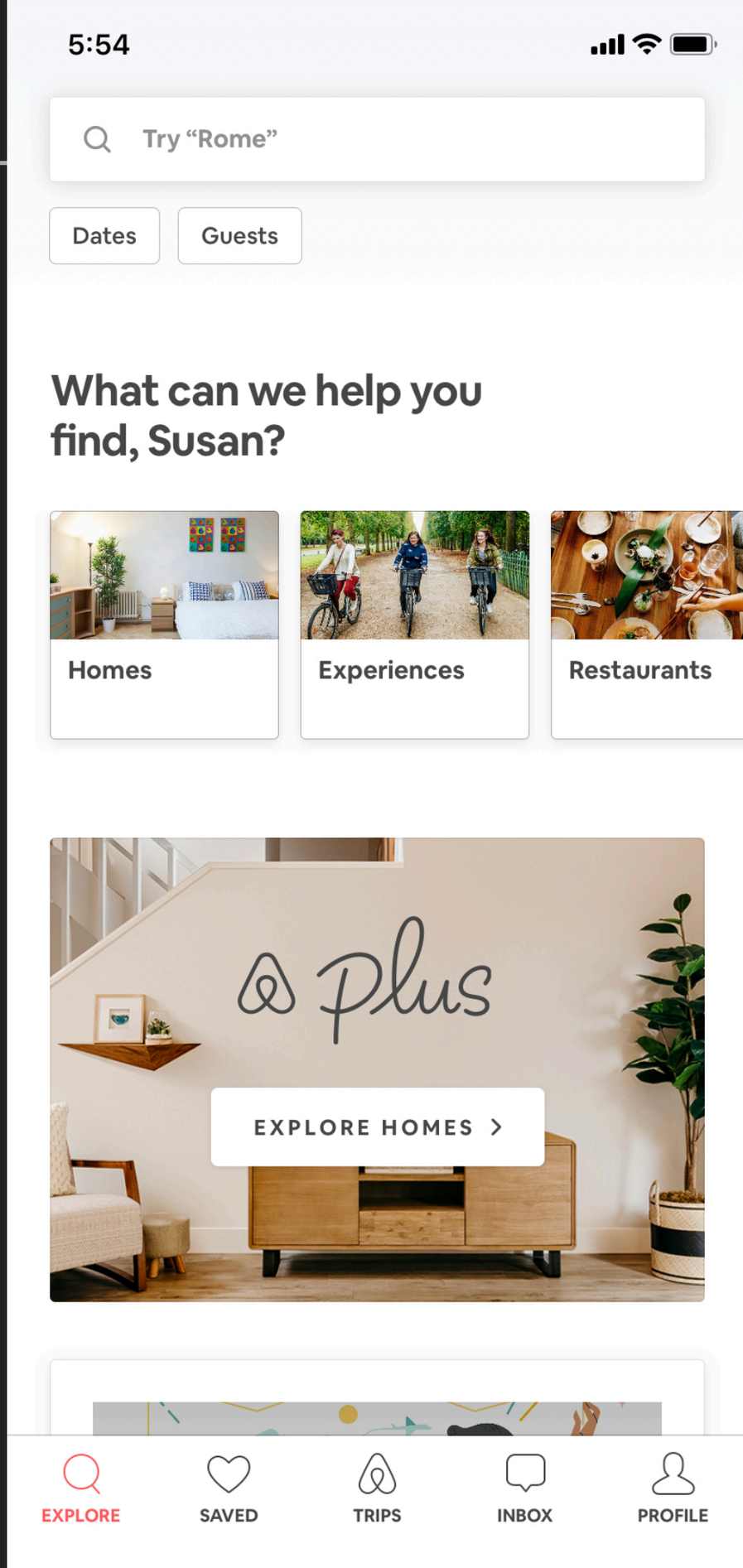
App Store



WHAT ABOUT...

- Shadows?

AirBnb



WHAT ABOUT...

- Gradients?

take a deep breath

Calm

. . . THESE ARE ALL PROPERTIES OF LAYERS

- Every UIView is backed by an instance of a class called `CALayer`
- Access a view's root layer with `view.layer`

LAYERS

- Layers are similar to views:
  - Geometry of a layer is defined by `frame` & `bounds`
  - Appearance of a layer can be modified with properties like `backgroundColor`
  - A layer can have a `superlayer` and `sublayers`

LAYERS

- Layers have a number of special properties that views do not, such as:
  - Corner radius
  - Border width and color
  - Shadow radius, offset and color

LAYERS

- There are a number of subclasses of CALayer, such as:
  - `CATextLayer` for rendering strings
  - `CAShapeLayer` for drawing shapes
  - `CAGradientLayer` for creating gradients

LAYERS

- Layers have many properties that can only be configured programmatically, not in storyboard
- A good place to do this is in `awakeFromNib`
  - This is a method on UIView
  - It can be overridden in a subclass
  - It’s called after a view is instantiated from storyboard

Simulator File Edit Hardware Debug Window Help 38% Jan 28 7:55 PM

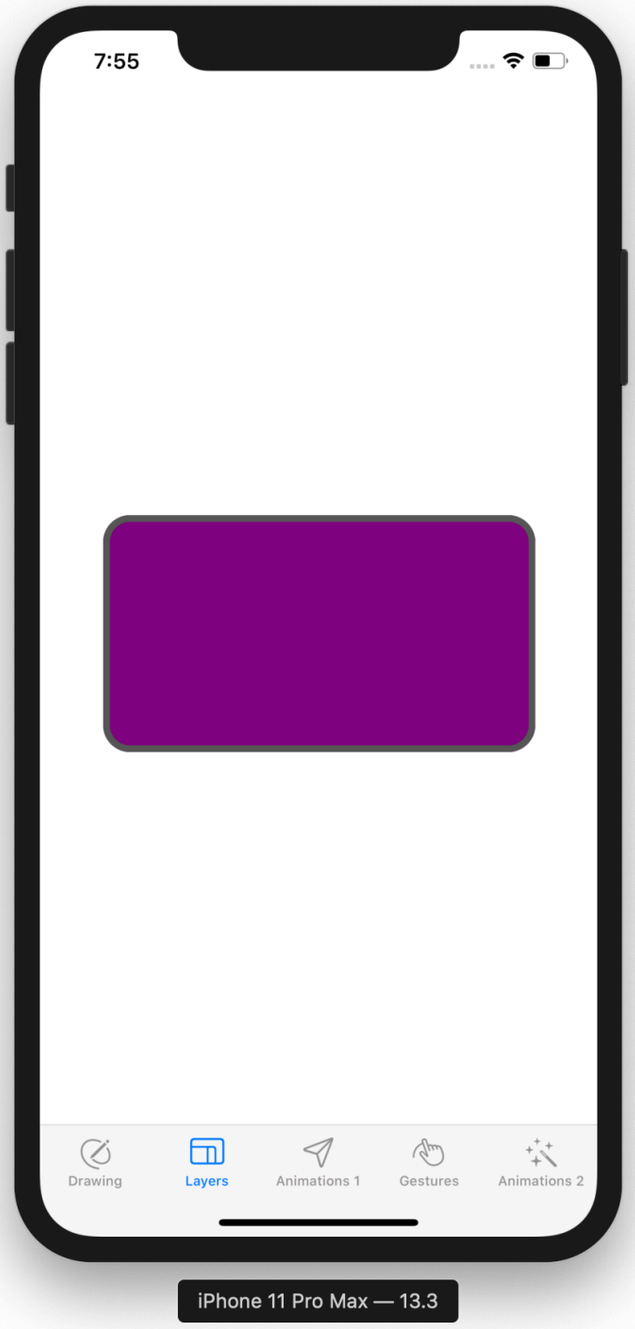
Session4Demos 〉 iPhone 11 Pro Max | Running Session4Demos on iPhone 11 Pro Max

Session4Demos 〉 Session4Demos 〉 Layers 〉 LayersView.swift 〉 No Selection

```
8
9   import UIKit
10
11  class LayersView: UIView {
12
13      override func awakeFromNib() {
14          super.awakeFromNib()
15
16          layer.backgroundColor = UIColor.purple.cgColor
17          layer.cornerRadius = 20
18          layer.borderColor = UIColor.darkGray.cgColor
19          layer.borderWidth = 5
20      }
21  }
22
23
24
25
26
27
28
29
30
31
32
33
```

Set cornerRadius to get rounded corners

Basic properties of CALayer



Session4Demos

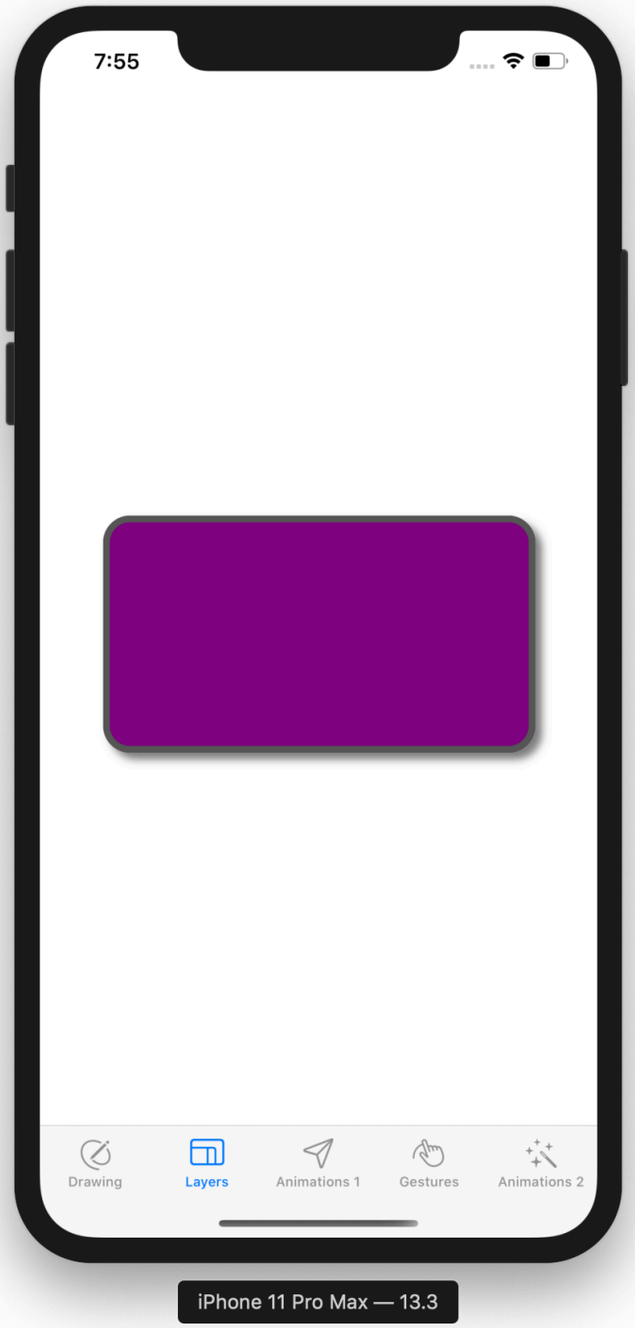
Simulator File Edit Hardware Debug Window Help 38% Jan 28 7:55 PM

Session4Demos 〉 iPhone 11 Pro Max | Running Session4Demos on iPhone 11 Pro Max

Session4Demos 〉 Session4Demos 〉 Layers 〉 LayersView.swift 〉 No Selection

```
8
9   import UIKit
10
11  class LayersView: UIView {
12
13      override func awakeFromNib() {
14          super.awakeFromNib()
15
16          layer.backgroundColor = UIColor.purple.cgColor
17          layer.cornerRadius = 20
18          layer.borderColor = UIColor.darkGray.cgColor
19          layer.borderWidth = 5
20
21          layer.shadowOpacity = 0.5
22          layer.shadowRadius = 3
23          layer.shadowOffset = CGSize(width: 5, height: 5)
24      }
25  }
26
27
28
29
30
31
32
33
```

Add a shadow



Session4Demos

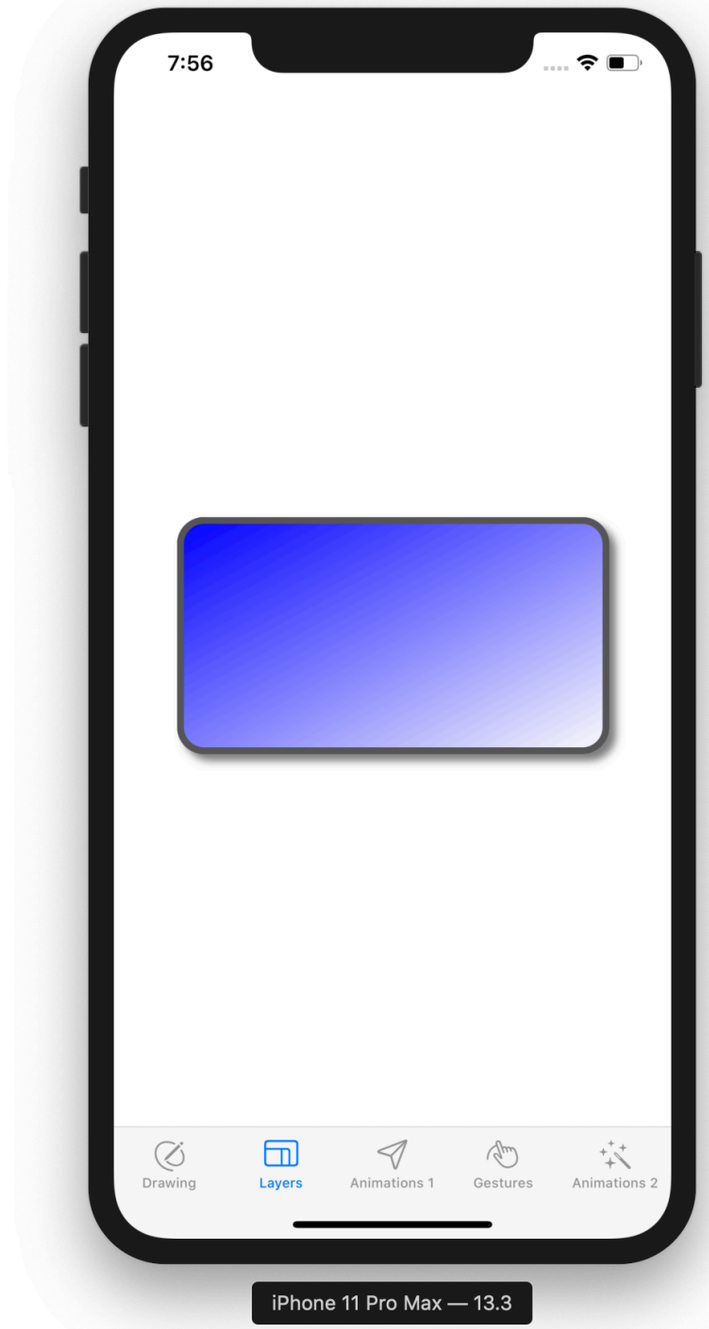
Simulator File Edit Hardware Debug Window Help 38% Jan 28 7:56 PM

Session4Demos 〉 iPhone 11 Pro Max | Running Session4Demos on iPhone 11 Pro Max

Session4Demos 〉 Session4Demos 〉 Layers 〉 LayersView.swift 〉 No Selection

```
10
11  class LayersView: UIView {
12
13      override func awakeFromNib() {
14          super.awakeFromNib()
15
16          layer.backgroundColor = UIColor.purple.cgColor
17          layer.cornerRadius = 20
18          layer.borderColor = UIColor.darkGray.cgColor
19          layer.borderWidth = 5
20
21          layer.shadowOpacity = 0.5
22          layer.shadowRadius = 3
23          layer.shadowOffset = CGSize(width: 5, height: 5)
24
25          let gradientLayer = CAGradientLayer()
26          gradientLayer.frame = bounds
27          gradientLayer.colors = [UIColor.blue.cgColor,
28                                  UIColor.white.cgColor]
29
30          gradientLayer.startPoint = CGPoint(x: 0, y: 0)
31          gradientLayer.endPoint = CGPoint(x: 1, y: 1)
32          gradientLayer.cornerRadius = 20
33          layer.addSublayer(gradientLayer)
34      }
35  }
```

Add a gradient sublayer



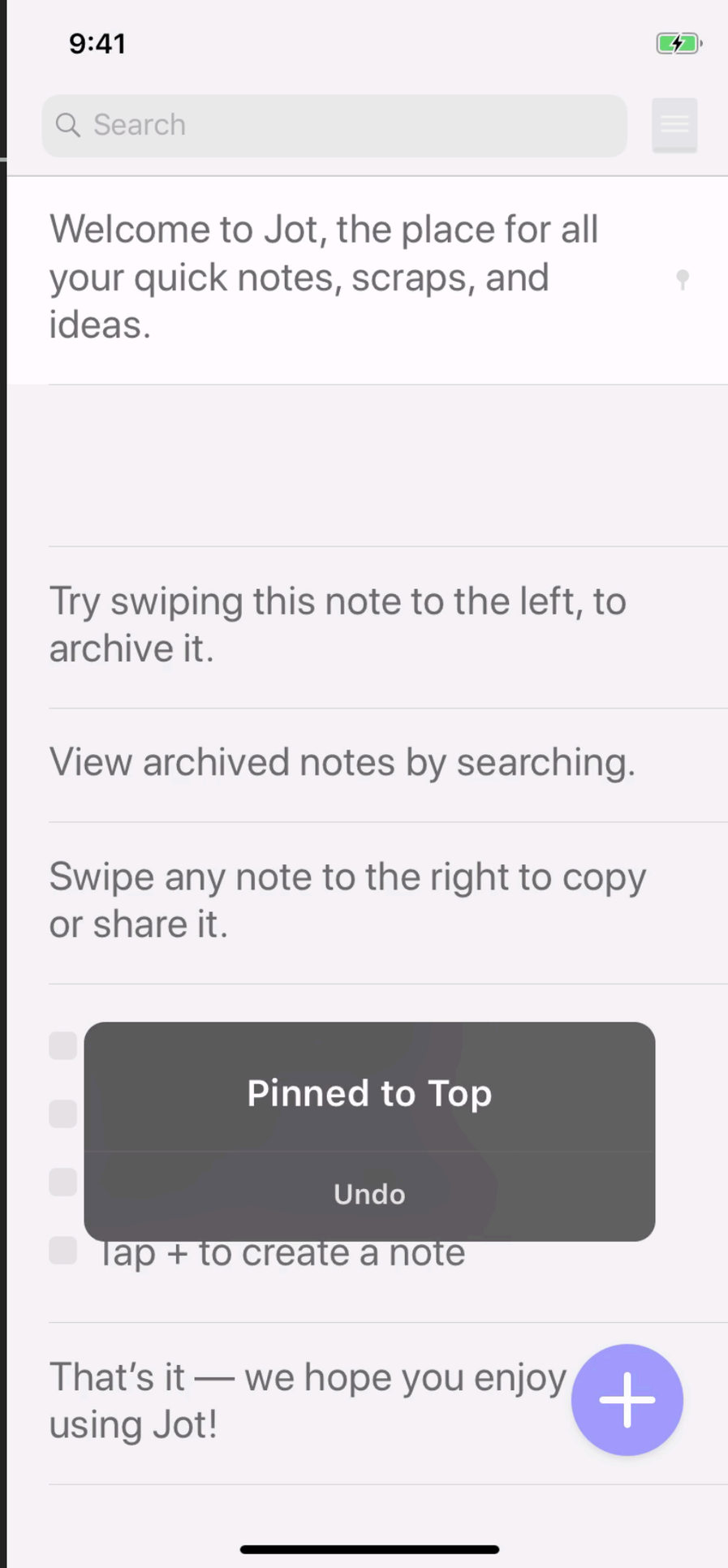
Session4Demos

PART 1

ANIMATIONS ✨

ANIMATION EXAMPLES

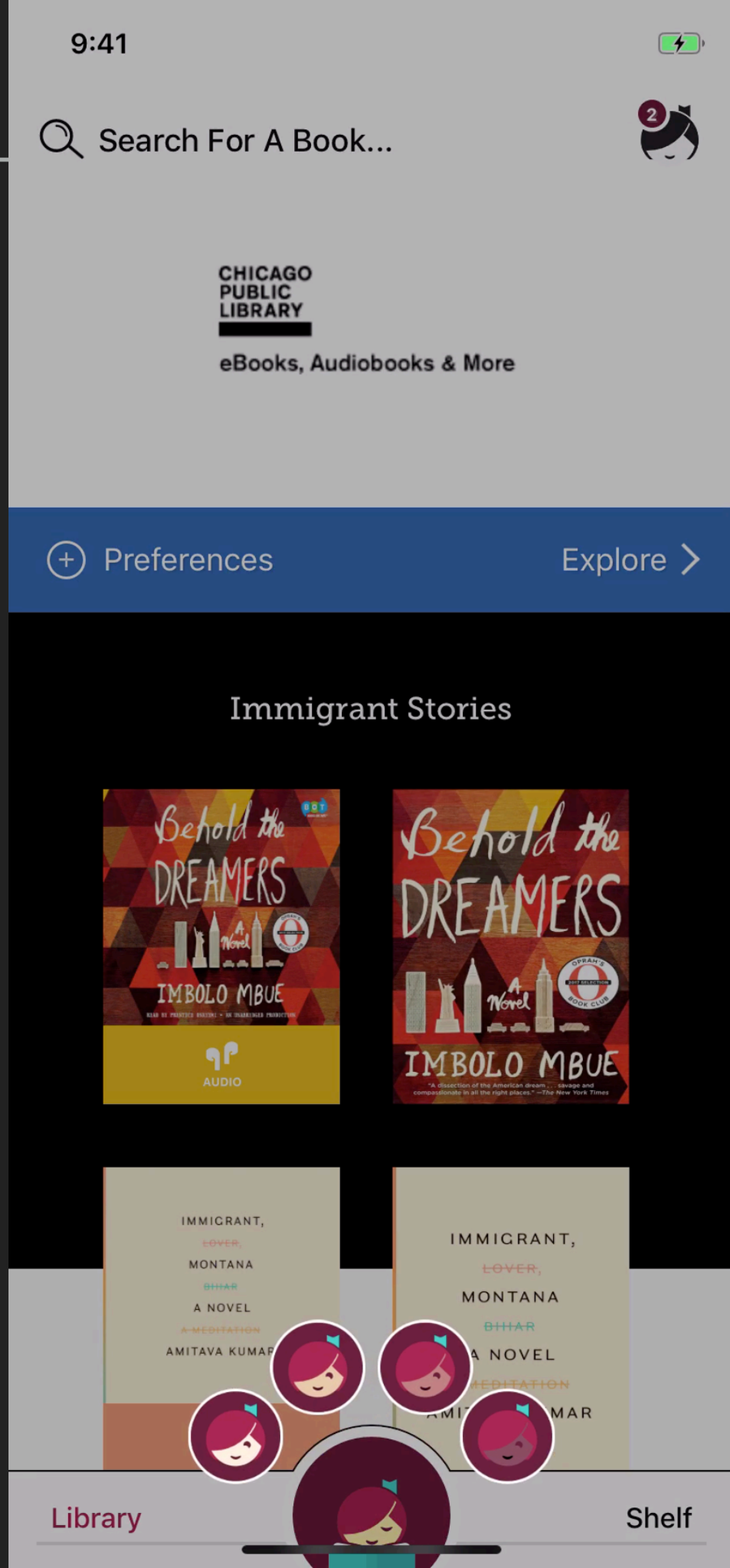
- Animate frame



ANIMATION EXAMPLES

- Animate alpha

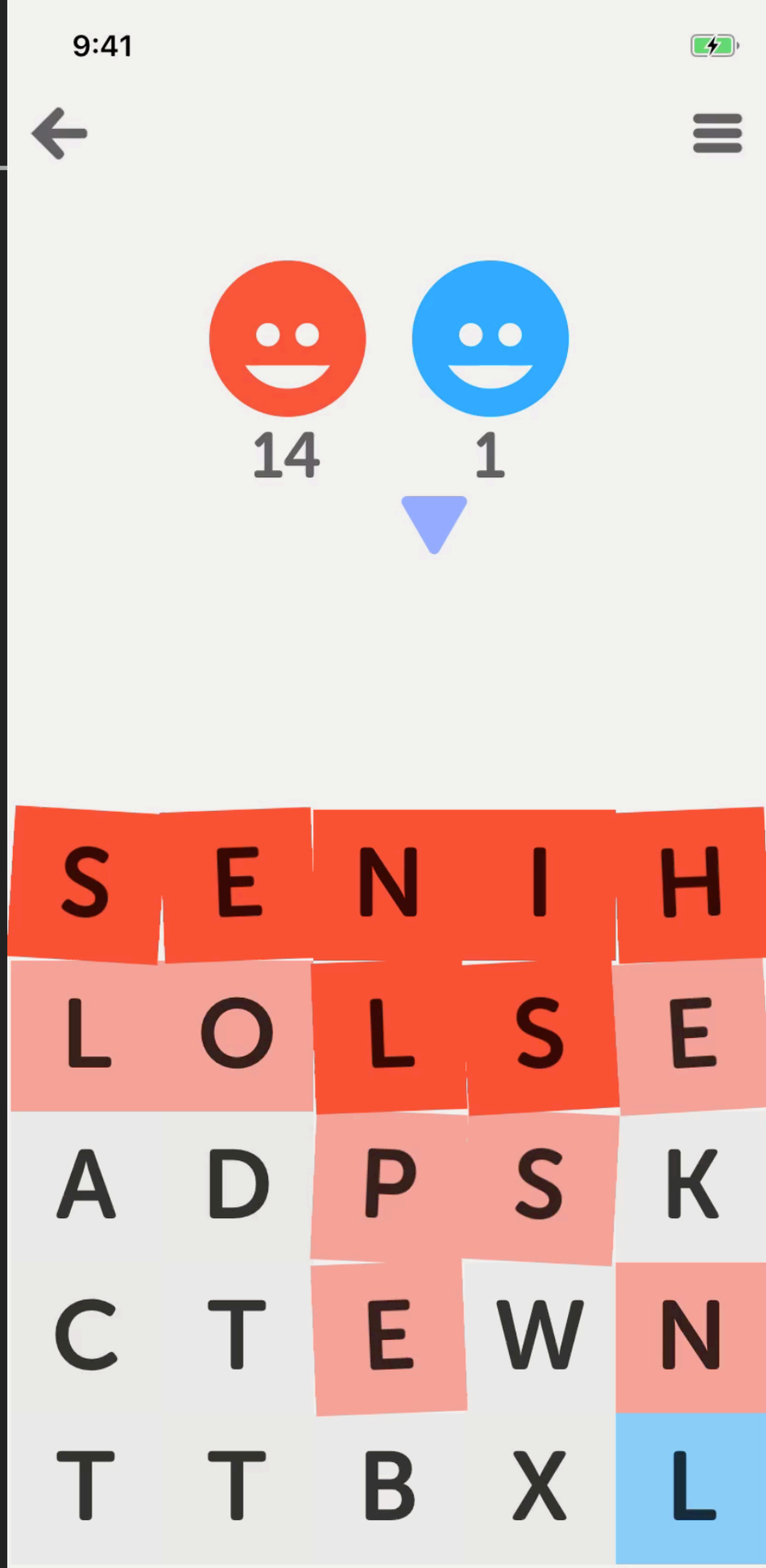
Libby



ANIMATION EXAMPLES

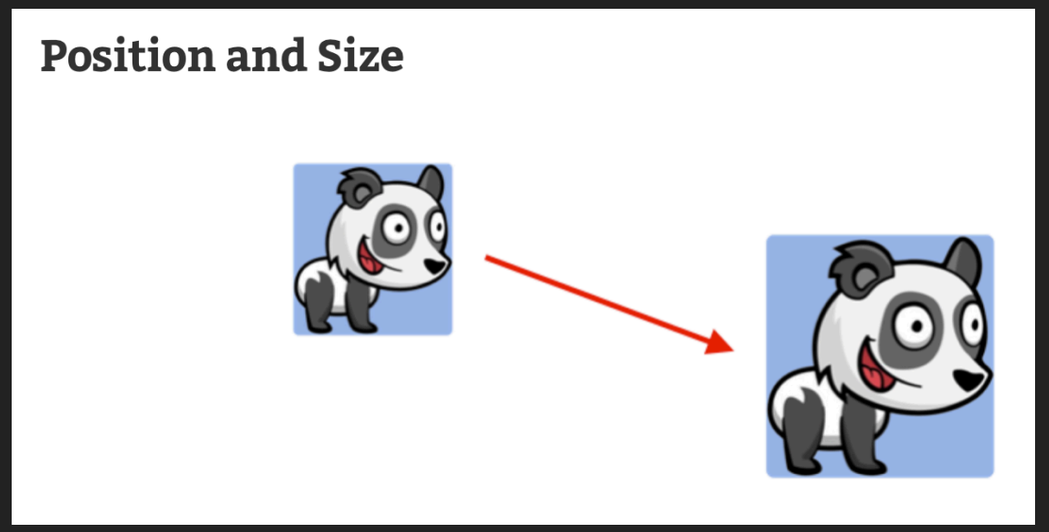
- Animate transform (rotation)

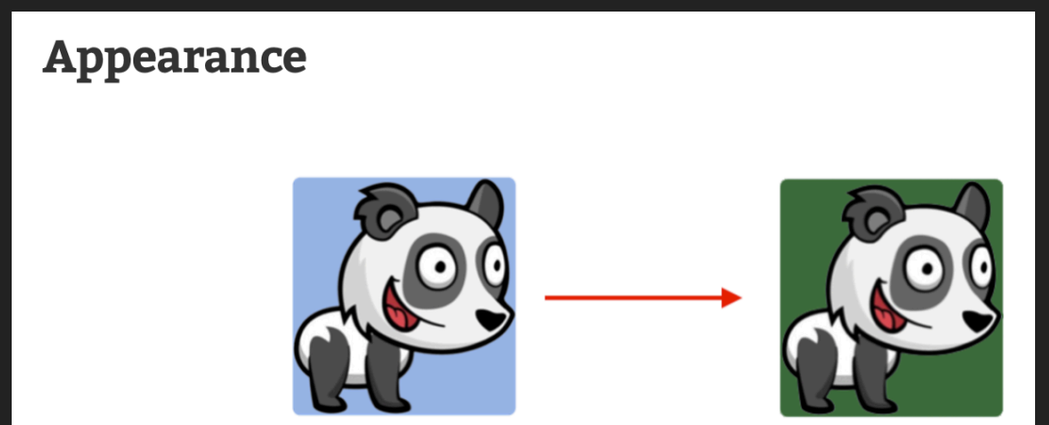
Letterpress

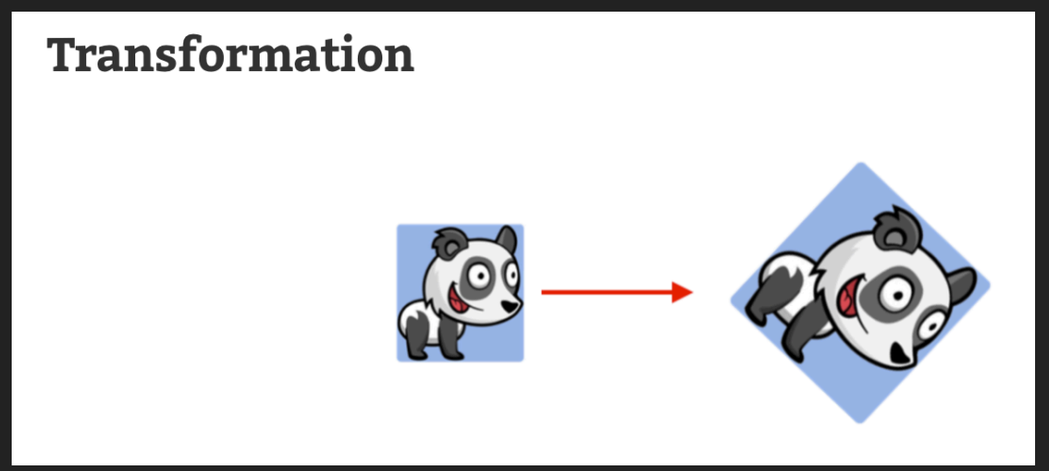


ANIMATABLE PROPERTIES

- Only certain properties of UIView can be animated:
  - Bounds, frame, center
  - Background color
  - Alpha
  - Transform (e.g., scale and rotation)





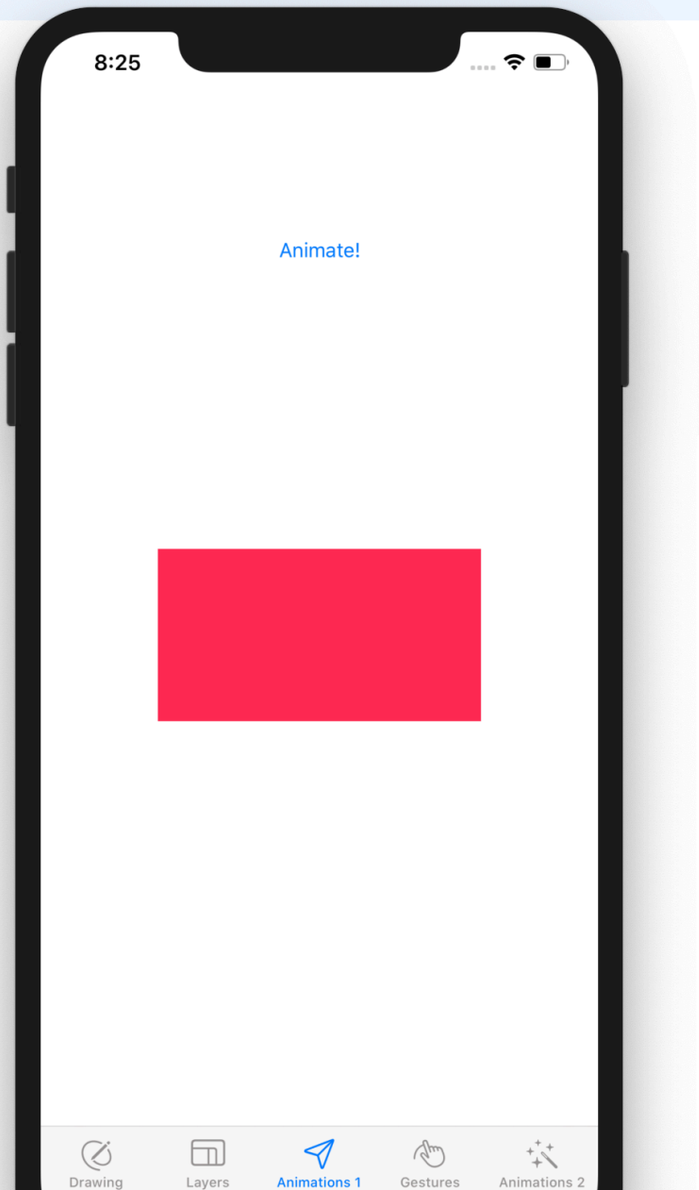


https://www.raywenderlich.com/363-ios-animation-tutorial-getting-started

SIMPLE ANIMATION

```
//

import UIKit

class Animations1ViewController: UIViewController {

    @IBOutlet var pinkView: UIView!

    @IBAction func tappedAnimate(_ sender: UIButton) {

        UIView.animate(withDuration: 1.0) {
            self.pinkView.alpha = 0.5
        }

    }
}
```

The `animate` function takes a TimeInterval and a closure



SIMPLE ANIMATION

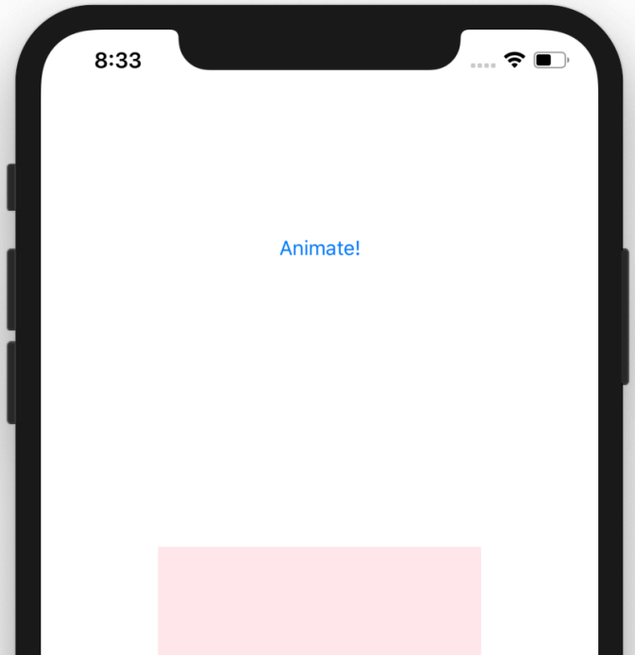
Simulator File Edit Hardware Debug Window Help 25% Jan 28 8:33 PM

Session4Demos 〉 iPhone 11 Pro Max | Running Session4Demos on iPhone 11 Pro Max | 5

Session4Demos 〉 Session4Demos 〉 Animations 1 〉 Animations1ViewController.swift 〉 No Selection

```
8
9   import UIKit
10
11  class Animations1ViewController: UIViewController {
12
13      @IBOutlet var pinkView: UIView!
14
15      @IBAction func tappedAnimate(_ sender: UIButton) {
16
17          UIView.animate(withDuration: 1.0, delay: 0.25, options: [.curveEaseIn], animations: {
18              self.pinkView.alpha = 0.1
19          }, completion: { _ in
20              print("animation complete")
21          })
22
23      }
24  }
25
```

Pass in additional parameters to further customize the animation



SIMPLE ANIMATION

- Additional parameters you can pass in:
    - Delay: how long to wait before starting the animation
    - Options: special options like repeat, autoreverse, and animation curves
    - Completion: closure to execute when the animation finishes

ANIMATION CURVES

- Specify whether the animation should slow down or speed up over the course of its run
- `.curveEaseInOut`: start slow, speed up, slow down
- `.curveEaseIn`: start slow, speed up, then stop
- `.curveEaseOut`: start fast, slow down, then stop
- `.curveLinear`: constant speed

PROPERTY ANIMATORS

- iOS 10 introduced `UIViewPropertyAnimator`
  - A new way of creating animations
  - The old way is “discouraged” but not deprecated… meaning, still widely in use

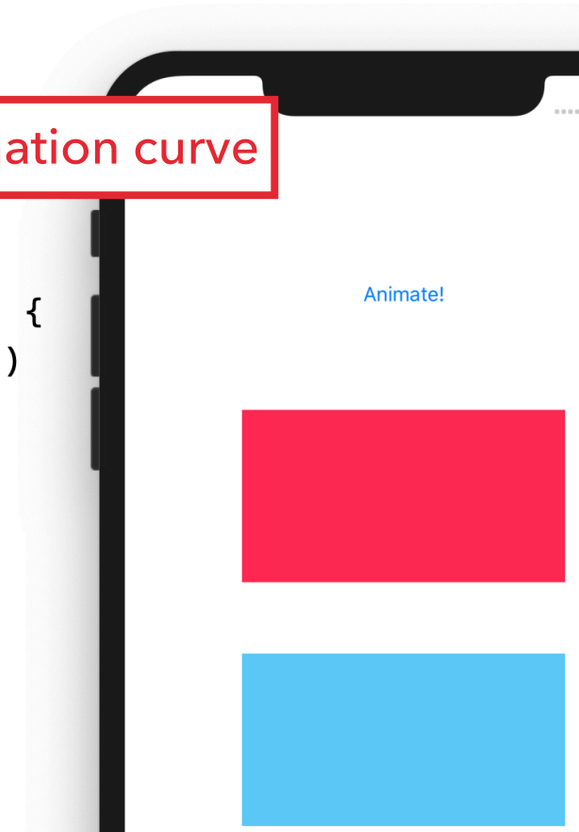
PROPERTY ANIMATORS

- `UIViewPropertyAnimator` allows fine-grained control over animations:
  - Pause and resume animations
  - Add additional animation blocks, even after an animation has begun
  - Modify the animation based on user interaction (swiping, dragging, etc.)

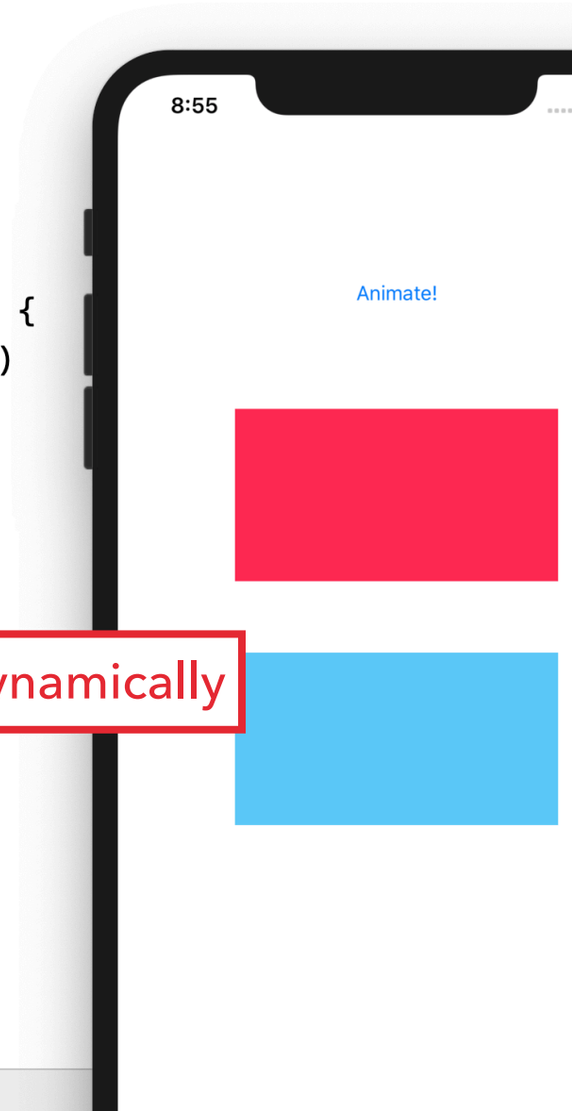
```swift
    @IBOutlet var pinkView: UIView!
    @IBOutlet var tealView: UIView!

    @IBAction func tappedAnimate(_ sender: UIButton) {

        // simple animation using UIView.animate
        UIView.animate(withDuration: 1.0, delay: 0.25, options: [.curveEaseIn], animations: {
            self.pinkView.alpha = 0.1
        }, completion: { _ in
            print("animation complete")
        })

        // animation using property animation
        let animator = UIViewPropertyAnimator(duration: 1.0, curve: .easeOut) {
            self.tealView.transform = CGAffineTransform(scaleX: 1.25, y: 1.25)
        }

        animator.addAnimations {
            self.tealView.backgroundColor = UIColor.systemPurple
        }

        animator.addCompletion { _ in
            self.tealView.transform = .identity
            self.tealView.backgroundColor = UIColor.systemTeal
        }

        animator.startAnimation()
    }
}
```

Duration in seconds

Animation curve



```swift
    @IBOutlet var pinkView: UIView!
    @IBOutlet var tealView: UIView!

    @IBAction func tappedAnimate(_ sender: UIButton) {

        // simple animation using UIView.animate
        UIView.animate(withDuration: 1.0, delay: 0.25, options: [.curveEaseIn], animations: {
            self.pinkView.alpha = 0.1
        }, completion: { _ in
            print("animation complete")
        })

        // animation using property animation
        let animator = UIViewPropertyAnimator(duration: 1.0, curve: .easeOut) {
            self.tealView.transform = CGAffineTransform(scaleX: 1.25, y: 1.25)
        }

        animator.addAnimations {
            self.tealView.backgroundColor = UIColor.systemPurple
        }

        animator.addCompletion { _ in
            self.tealView.transform = .identity
            self.tealView.backgroundColor = UIColor.systemTeal
        }

        animator.startAnimation()
    }
}
```



Add more animations dynamically

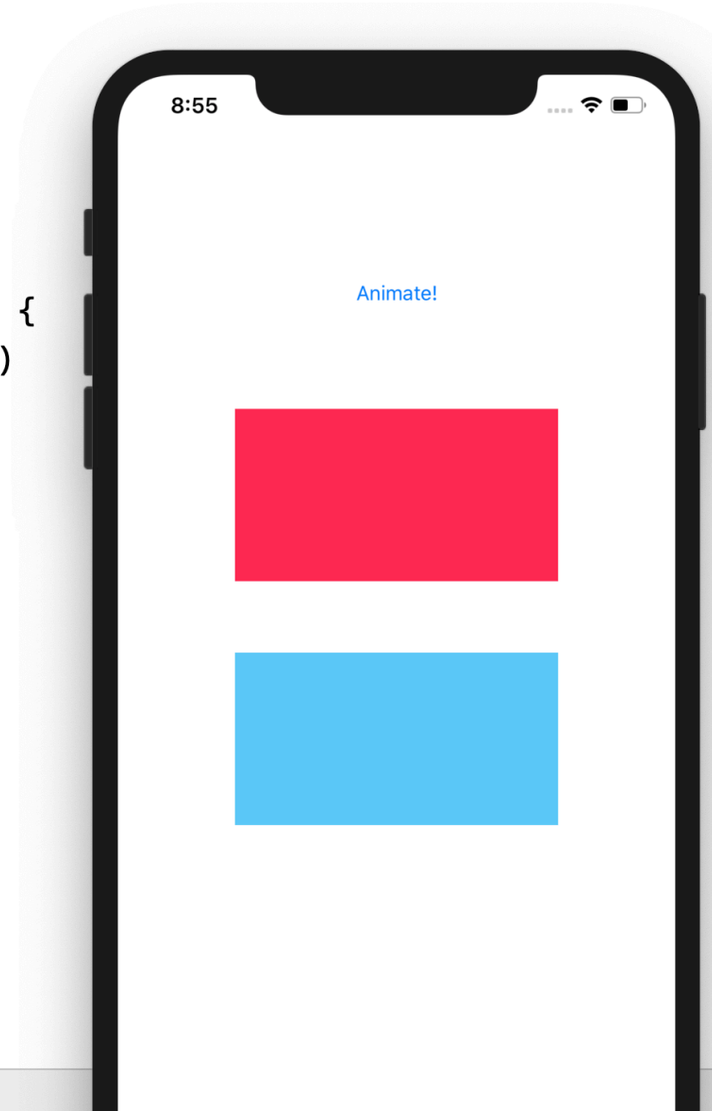
Simulator File Edit Hardware Debug Window Help 19% Jan 28 8:55 PM

Session4Demos 〉 iPhone 11 Pro Max | Running Session4Demos on iPhone 11 Pro Max | 5

Session4Demos 〉 Session4Demos 〉 Animations 1 〉 Animations1ViewController.swift 〉 No Selection

```
    @IBOutlet var pinkView: UIView!
    @IBOutlet var tealView: UIView!

    @IBAction func tappedAnimate(_ sender: UIButton) {

        // simple animation using UIView.animate
        UIView.animate(withDuration: 1.0, delay: 0.25, options: [.curveEaseIn], animations: {
            self.pinkView.alpha = 0.1
        }, completion: { _ in
            print("animation complete")
        })

        // animation using property animation
        let animator = UIViewPropertyAnimator(duration: 1.0, curve: .easeOut) {
            self.tealView.transform = CGAffineTransform(scaleX: 1.25, y: 1.25)
        }

        animator.addAnimations {
            self.tealView.backgroundColor = UIColor.systemPurple
        }

        animator.addCompletion { _ in
            self.tealView.transform = .identity
            self.tealView.backgroundColor = UIColor.systemTeal
        }

        animator.startAnimation()
    }
}
```

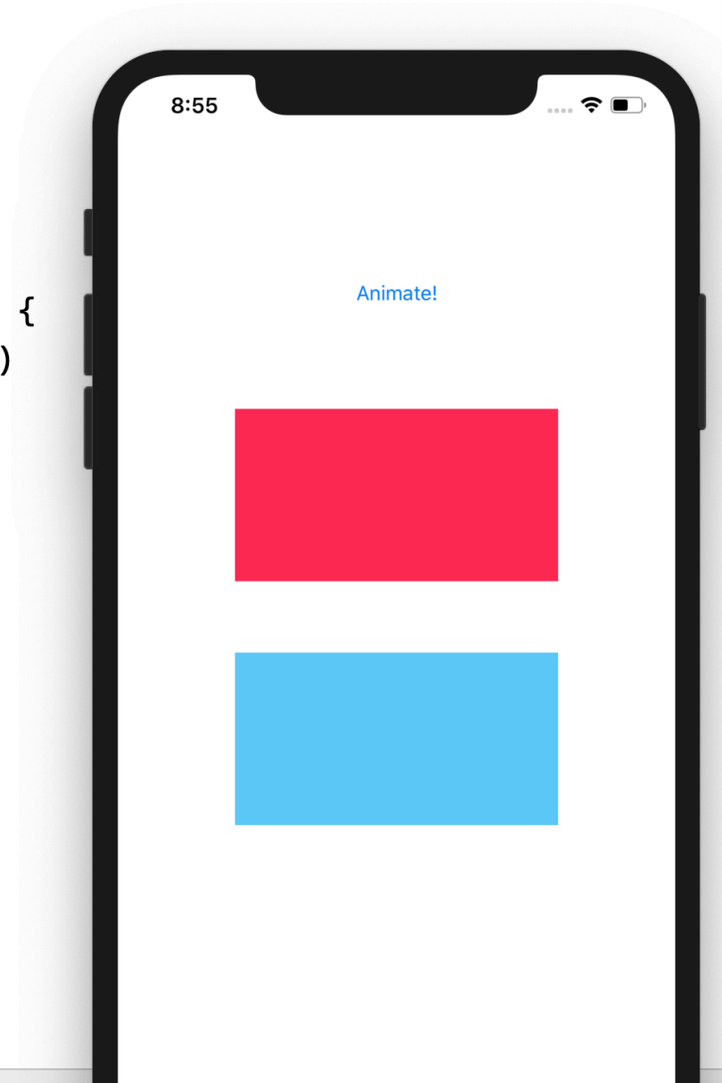
Add a completion



Session4Demos

```swift
    @IBOutlet var pinkView: UIView!
    @IBOutlet var tealView: UIView!

    @IBAction func tappedAnimate(_ sender: UIButton) {

        // simple animation using UIView.animate
        UIView.animate(withDuration: 1.0, delay: 0.25, options: [.curveEaseIn], animations: {
            self.pinkView.alpha = 0.1
        }, completion: { _ in
            print("animation complete")
        })

        // animation using property animation
        let animator = UIViewPropertyAnimator(duration: 1.0, curve: .easeOut) {
            self.tealView.transform = CGAffineTransform(scaleX: 1.25, y: 1.25)
        }

        animator.addAnimations {
            self.tealView.backgroundColor = UIColor.systemPurple
        }

        animator.addCompletion { _ in
            self.tealView.transform = .identity
            self.tealView.backgroundColor = UIColor.systemTeal
        }

        animator.startAnimation()
    }
}
```

Tell the animator to start



ADDING USER INTERACTION

GESTURE RECOGNIZERS

GESTURE RECOGNIZERS

- Gesture recognizers are objects that allow your code to detect and respond to common gestures:
  - Tap
  - Pinch
  - Rotation
  - Swipe
  - Pan
  - Screen edge pan
  - Long press

GESTURE RECOGNIZERS

- `UIGestureRecognizer` is a base class that has concrete subclasses for each possible gesture

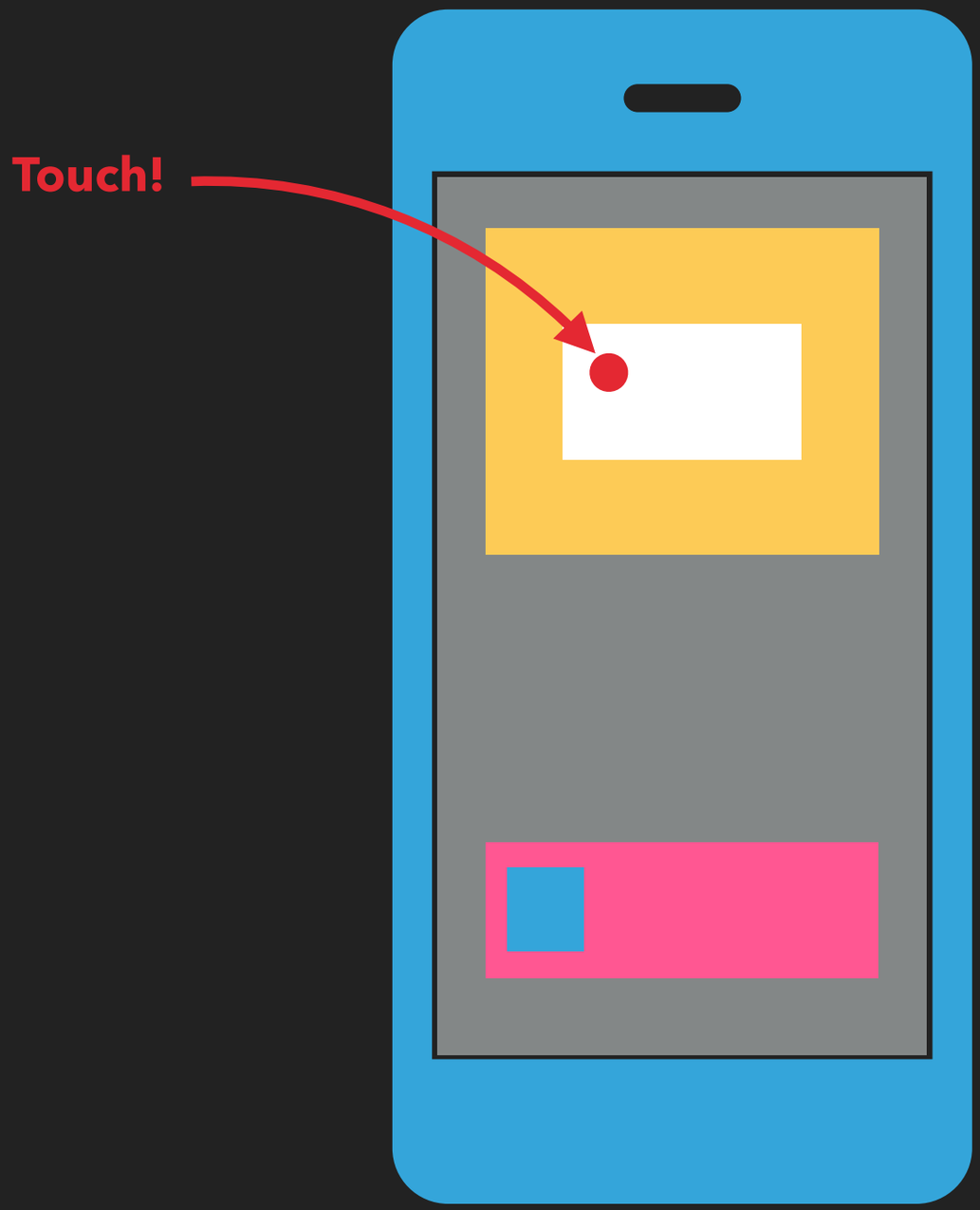
The concrete subclasses of `UIGestureRecognizer` are the following:

- `UITapGestureRecognizer`
- `UIPinchGestureRecognizer`
- `UIRotationGestureRecognizer`
- `UISwipeGestureRecognizer`
- `UIPanGestureRecognizer`
- `UIScreenEdgePanGestureRecognizer`
- `UILongPressGestureRecognizer`

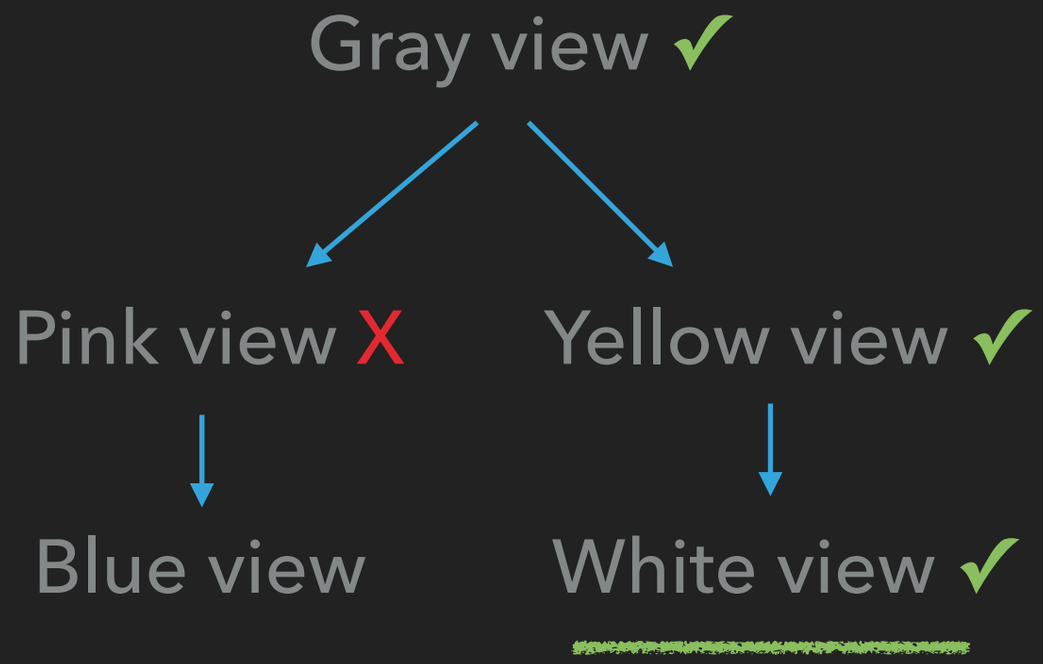
GESTURE RECOGNIZERS

- Hit-testing is used to determine which view should receive a touch event from the user
  - The user touches the screen at a particular point
  - The system traverses the view hierarchy, starting with the root view, looking for the front-most view containing the point

GESTURE RECOGNIZERS



EXAMPLE: HIT-TESTING VIEWS



GESTURE RECOGNIZERS

- Each gesture recognizer you create must be associated with a specific view
    - The gesture recognizer responds to touch events for that view and its subviews
- If a view receives a touch event that it can’t respond to, the system will traverse down the view hierarchy looking for a superview to handle the event

GESTURE RECOGNIZERS

- Gesture recognizers are initialized with a target and an action (selector)

```
11 class GesturesViewController: UIViewController {
12
13     override func viewDidLoad() {
14         super.viewDidLoad()
15
16         let tapGestureRecognizer = UITapGestureRecognizer(target: self,
17                                                           action: #selector(tap))
18         tapGestureRecognizer.numberOfTapsRequired = 2
19         view.addGestureRecognizer(tapGestureRecognizer)
20     }
21
```

GESTURE RECOGNIZERS

- Call `addGestureRecognizer` to associate the gesture recognizer with a specific view

```
11 class GesturesViewController: UIViewController {
12
13     override func viewDidLoad() {
14         super.viewDidLoad()
15
16         let tapGestureRecognizer = UITapGestureRecognizer(target: self,
17                                                           action: #selector(tap))
18         tapGestureRecognizer.numberOfTapsRequired = 2
19         view.addGestureRecognizer(tapGestureRecognizer)
20     }
21
```

GESTURE RECOGNIZERS

- Gesture recognizers also have configurable properties, depending on the type of gesture

```
11  class GesturesViewController: UIViewController {
12
13      override func viewDidLoad() {
14          super.viewDidLoad()
15
16          let tapGestureRecognizer = UITapGestureRecognizer(target: self,
17                                                            action: #selector(tap))
            tapGestureRecognizer.numberOfTapsRequired = 2
18          view.addGestureRecognizer(tapGestureRecognizer)
20      }
21
```

Only recognize double-taps

TAP GESTURE RECOGNIZER

Configuring the Gesture

```
var numberOfTapsRequired: Int
```

The number of taps for the gesture to be recognized.

```
var numberOfTouchesRequired: Int
```

The number of fingers required to tap for the gesture to be recognized.

https://developer.apple.com/documentation/uikit/uitapgesturerecognizer

SWIPE GESTURE RECOGNIZER

Configuring the Gesture

```
var direction: UISwipeGestureRecognizer.Direction
```

The permitted direction of the swipe for this gesture recognizer.

```
var numberOfTouchesRequired: Int
```

The number of touches that must be present for the swipe gesture to be recognized.

https://developer.apple.com/documentation/uikit/uiswipegesturerecognizer

PAN GESTURE RECOGNIZER

Configuring the Gesture Recognizer

`var maximumNumberOfTouches: Int`

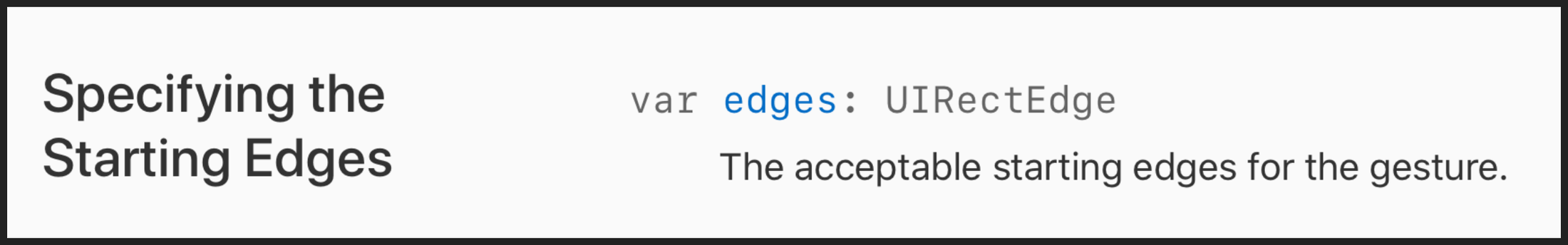
The maximum number of fingers that can be touching the view for this gesture to be recognized.

`var minimumNumberOfTouches: Int`

The minimum number of fingers that can be touching the view for this gesture to be recognized.

https://developer.apple.com/documentation/uikit/uipangesturerecognizer

SCREEN EDGE PAN GESTURE RECOGNIZER



https://developer.apple.com/documentation/uikit/uiscreenedgepangesturerecognizer

LONG PRESS GESTURE RECOGNIZER

Configuring the Gesture Recognizer

`var minimumPressDuration: TimeInterval`

The minimum period fingers must press on the view for the gesture to be recognized.

`var numberOfTouchesRequired: Int`

The number of fingers that must be pressed on the view for the gesture to be recognized.

`var numberOfTapsRequired: Int`

The number of taps on the view required for the gesture to be recognized.

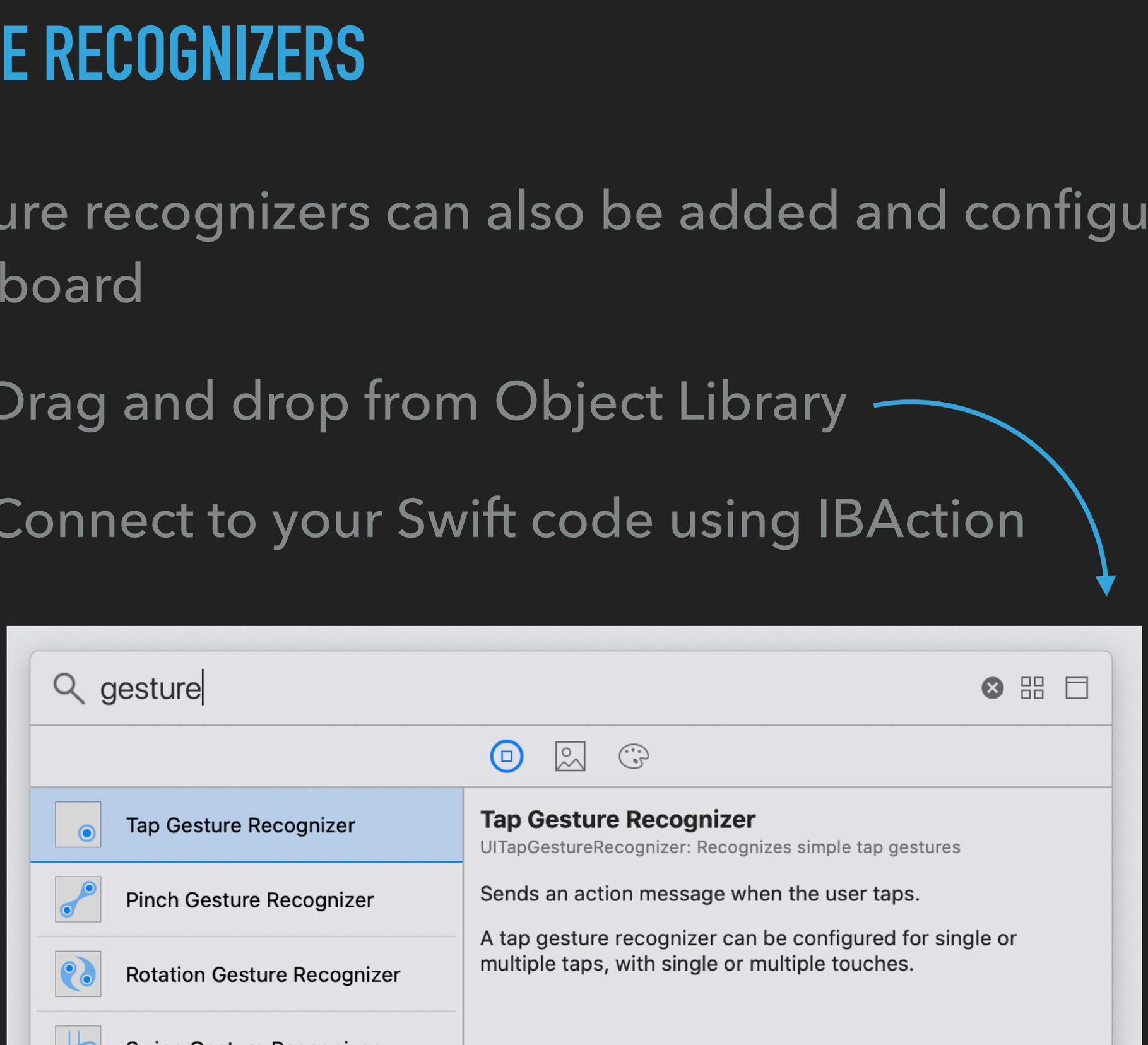
`var allowableMovement: CGFloat`

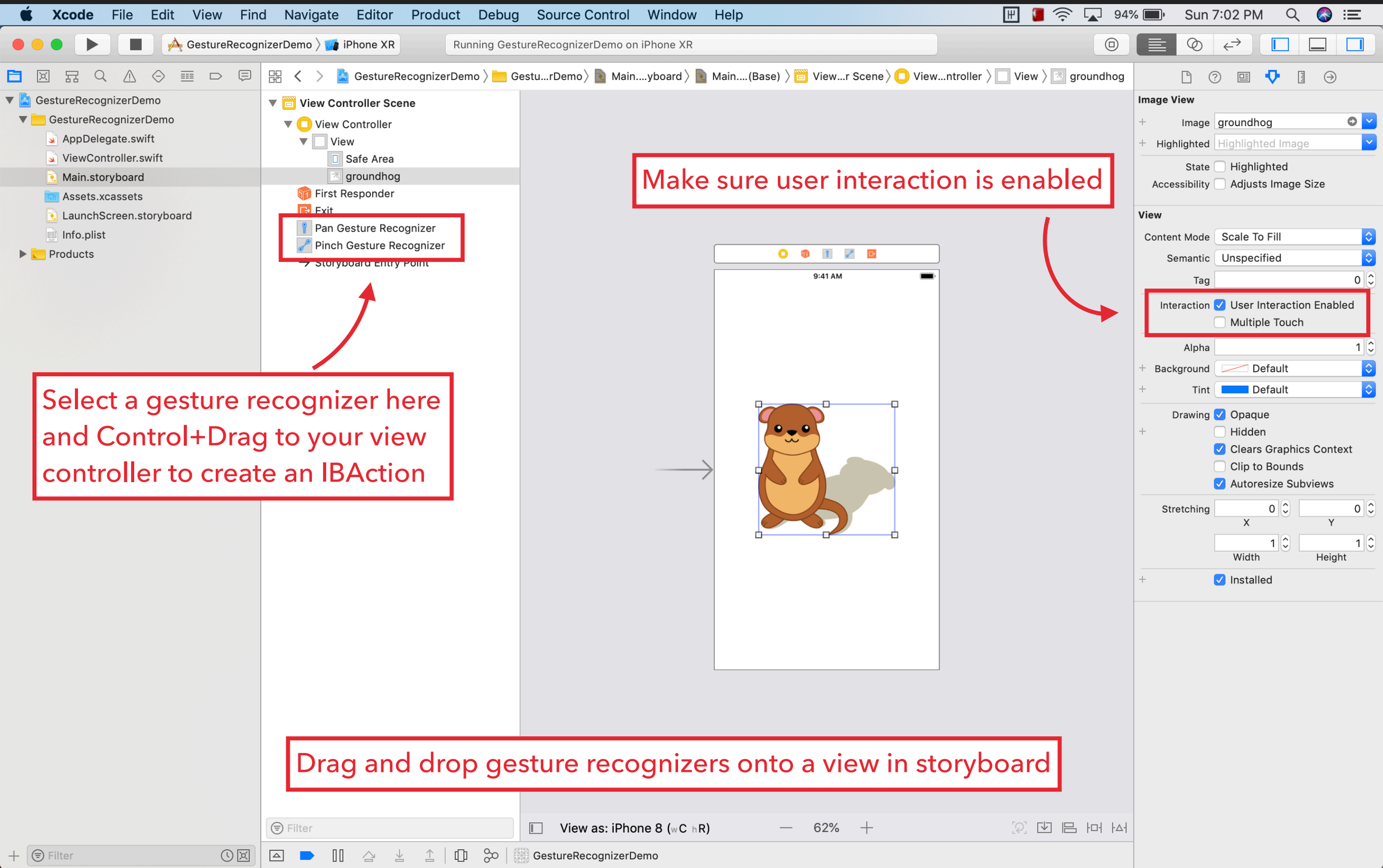
The maximum movement of the fingers on the view before the gesture fails.

https://developer.apple.com/documentation/uikit/uilongpressgesturerecognizer

GESTURE RECOGNIZERS

- Gesture recognizers can also be added and configured in storyboard
  - Drag and drop from Object Library
  - Connect to your Swift code using IBAction




Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
94% Sun 7:02 PM
GestureRecognizerDemo iPhone XR
Running GestureRecognizerDemo on iPhone XR
GestureRecognizerDemo
GestureRecognizerDemo
AppDelegate.swift
ViewController.swift
Main.storyboard
Assets.xcassets
LaunchScreen.storyboard
Info.plist
Products
View Controller Scene
View Controller
View
Safe Area
groundhog
First Responder
Exit
Pan Gesture Recognizer
Pinch Gesture Recognizer
Storyboard Entry Point
Make sure user interaction is enabled
9:41 AM
Image View
Image groundhog
Highlighted Highlighted Image
State Highlighted
Accessibility Adjusts Image Size
View
Content Mode Scale To Fill
Semantic Unspecified
Tag 0
Interaction User Interaction Enabled
Multiple Touch
Alpha 1
Background Default
Tint Default
Drawing Opaque
Hidden
Clears Graphics Context
Clip to Bounds
Autoresize Subviews
Stretching 0 X 0 Y
1 Width 1 Height
Installed
Select a gesture recognizer here and Control+Drag to your view controller to create an IBAction
Drag and drop gesture recognizers onto a view in storyboard
View as: iPhone 8 (wC hR) 62%
GestureRecognizerDemo

GESTURE RECOGNIZERS

- Hook up gesture recognizers to code with IBActions

```
9  import UIKit
10
11 class ViewController: UIViewController {
12
13     @IBAction func pan(_ sender: UIPanGestureRecognizer) {
14         print("Pan!")
15     }
16
17     @IBAction func pinch(_ sender: UIPinchGestureRecognizer) {
18         print("Pinch!")
19     }
20 }
21
```

HANDLING GESTURE RECOGNIZERS

- Many gesture are continuous, meaning they take place over a period of time
- The action method will be called multiple times
- You can inspect properties of the gesture recognizer to determine how to respond each time the method is called

GESTURE RECOGNIZERS

- You may want to update your view based on the location of the touch or the number of touches

Getting the Touches and Location of a Gesture

```
func location(in: UIView?) -> CGPoint
```

Returns the point computed as the location in a given view of the gesture represented by the receiver.

```
func location(ofTouch: Int, in: UIView?) -> CGPoint
```

Returns the location of one of the gesture's touches in the local coordinate system of a given view.

```
var numberOfTouches: Int
```

Returns the number of touches involved in the gesture represented by the receiver.

GESTURE RECOGNIZERS

- You may also want to know the state of the gesture recognizer, using `gestureRecognizer.state`
- Possible values include:
  - Began
  - Changed
  - Ended
  - Possible

GESTURE RECOGNIZERS

- Specific subclasses of UIGestureRecognizer will also have their own properties
- For example:
  - Pan – translation and velocity
  - Pinch – scale and velocity

SIMULATOR TESTING

- How do you simulate gestures that require two fingers?
  - Hold down Option to get two “fingers” on simulator
  - Hold down Option+Shift to move both “fingers” together

PART 2

ANIMATIONS ✨

BUILDING MORE COMPLEX ANIMATIONS

- We have seen how to animate properties of `UIView`
- You can also animate properties of `CALayer` to create more complex animations
- `CALayer` is part of the Core Animation framework

BUILDING MORE COMPLEX ANIMATIONS

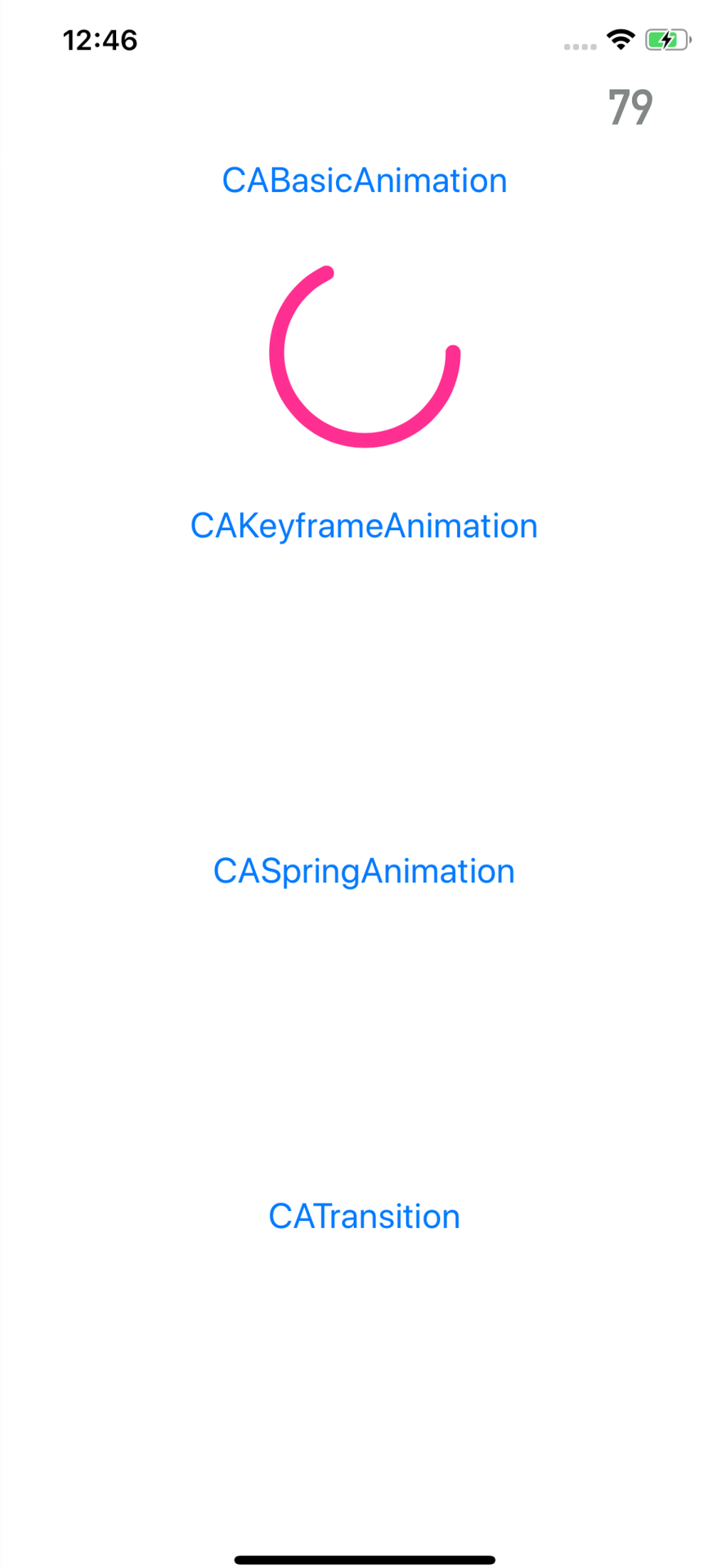
- See CoreAnimationDemo project in class repo for code samples

BUILDING MORE COMPLEX ANIMATIONS

- Subclasses of CALayer
  - `CAGradientLayer`
  - `CAShapeLayer`
  - `CATextLayer`

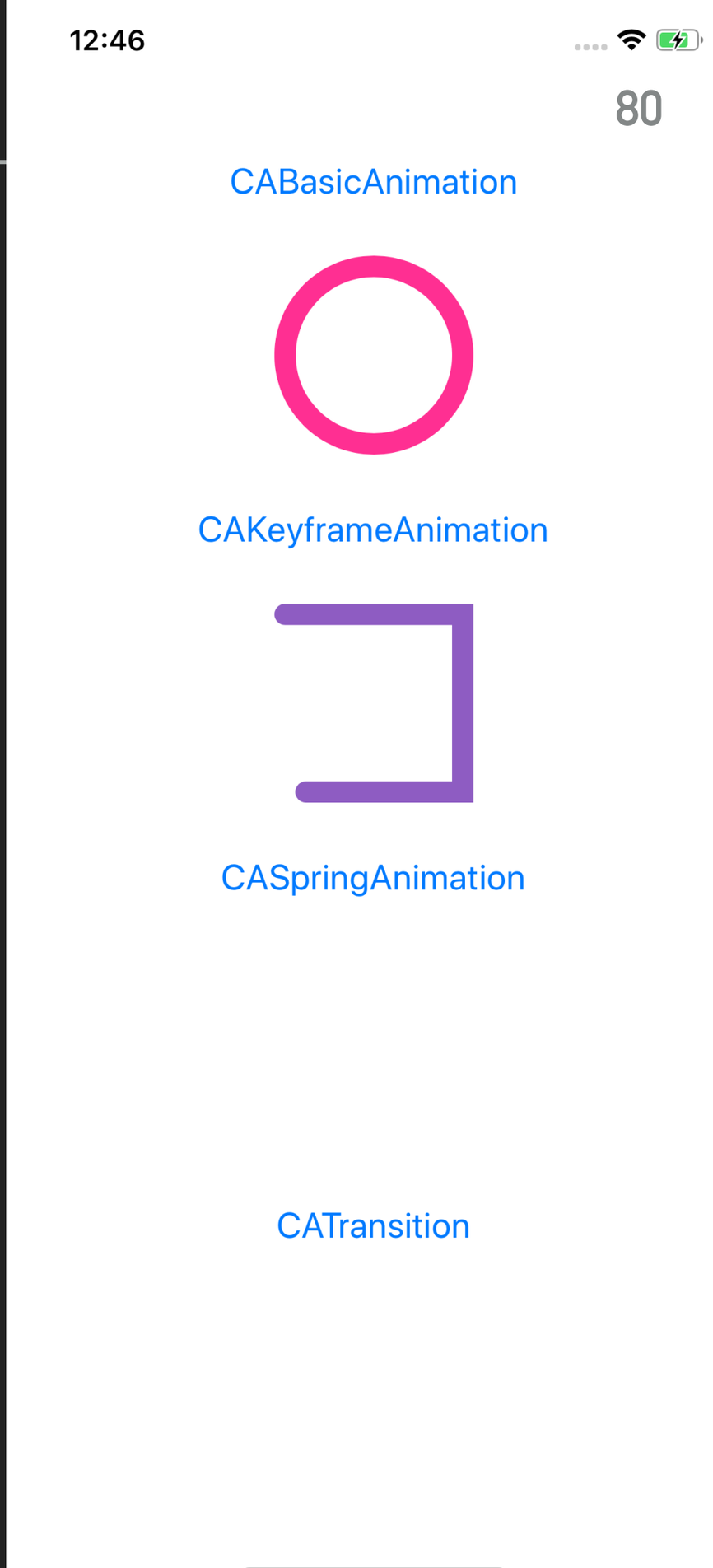
BUILDING MORE COMPLEX ANIMATIONS

- `CABasicAnimation`
  - Animate a property from one value to another
- E.g., use `strokeEnd` to animate drawing a path



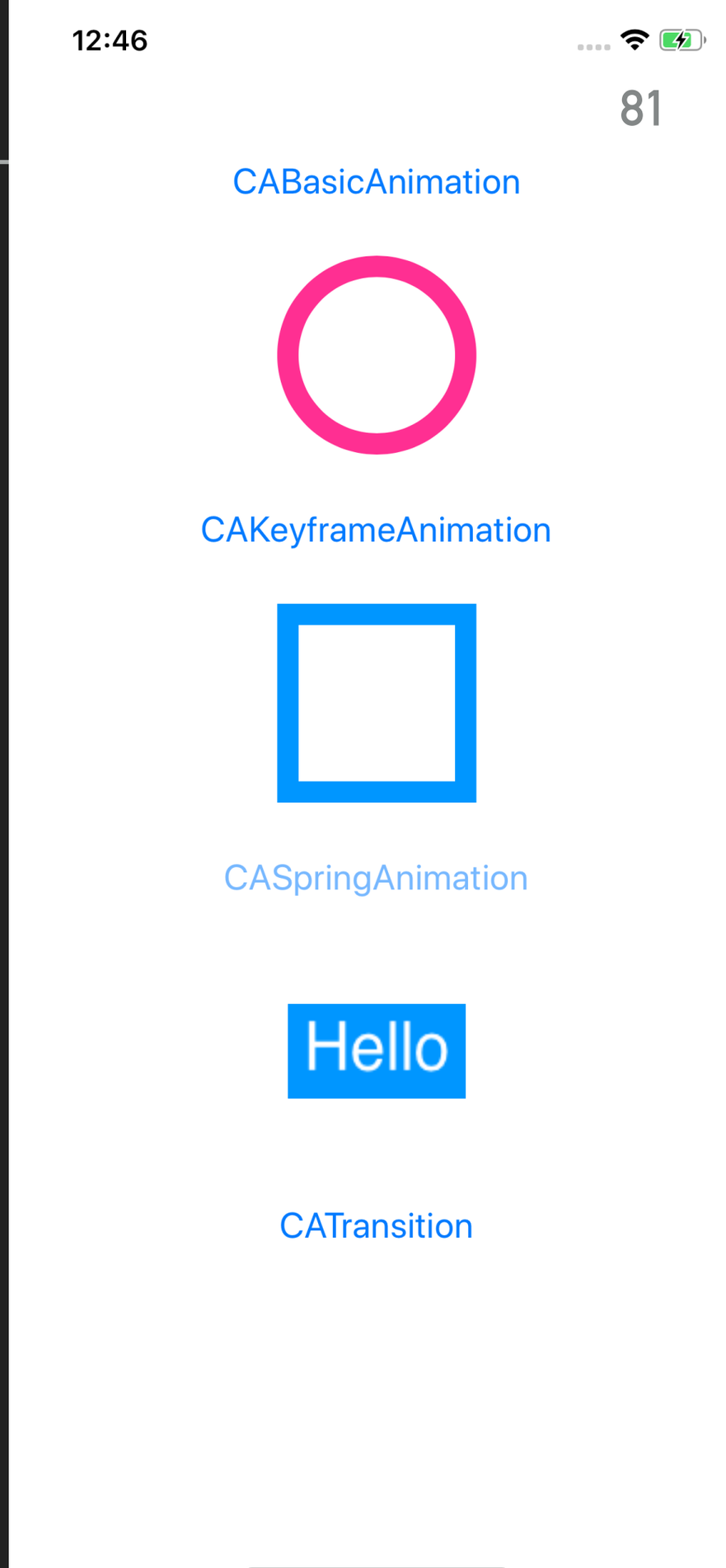
BUILDING MORE COMPLEX ANIMATIONS

- `CAKeyframeAnimation`
  - Animate a property between multiple values
- E.g., animate `strokeColor` from pink to purple to blue



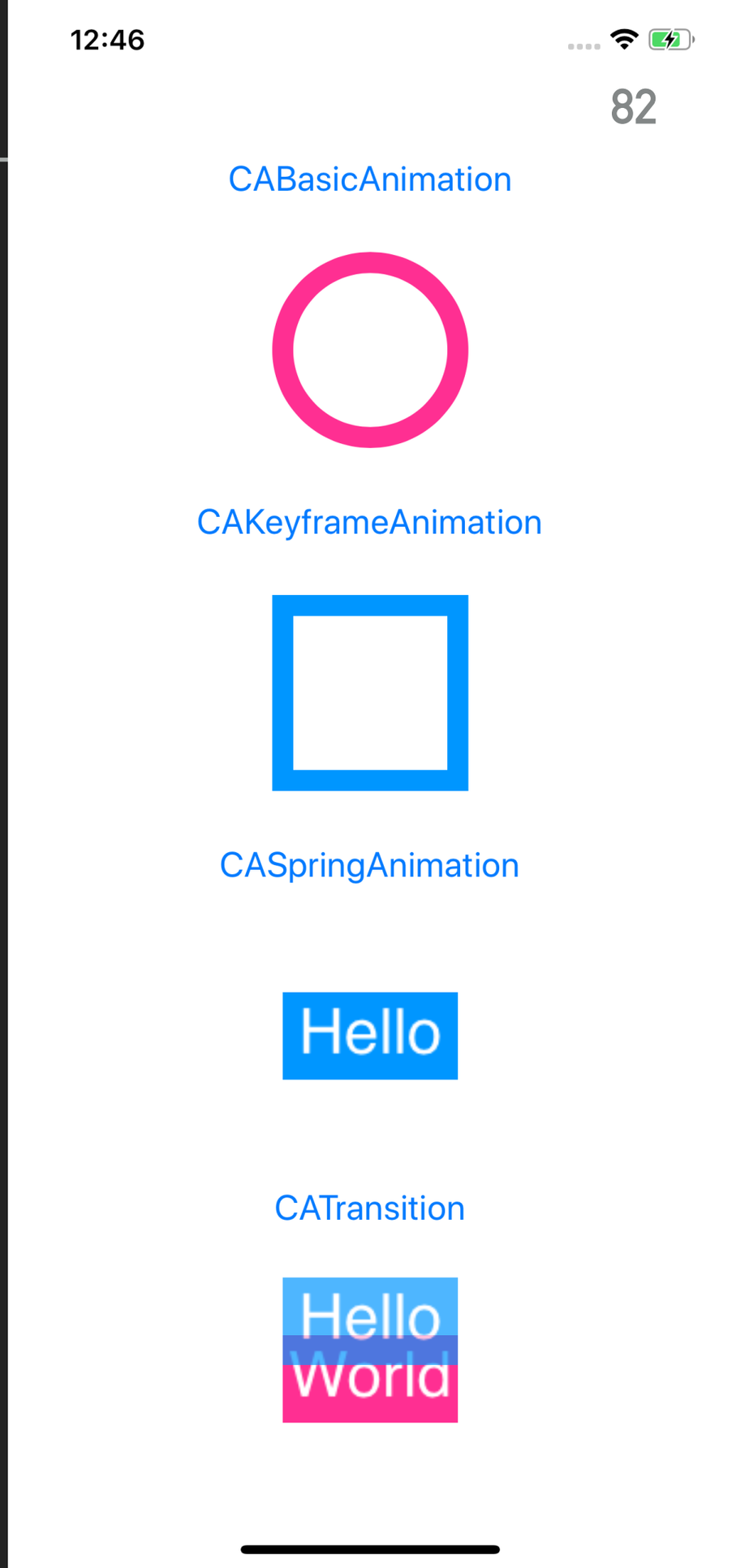
BUILDING MORE COMPLEX ANIMATIONS

- `CASpringAnimation`
  - Animate a property with a spring (bounce) effect
  - E.g., use spring animation on `transform.scale.y`



BUILDING MORE COMPLEX ANIMATIONS

- `CATransition`
  - Animate a layer with a given transition style
  - E.g., initial state floats to the top as a new state appears



ASSIGNMENT #4

TIC TAC TOE

TIC TAC TOE

- Build a Tic Tac Toe game for two players to play against each other on a single device
- You will use:
  - `UIBezierPath`
  - `CALayers`
  - `UIView` animation
  - `UIGestureRecognizers`

TIC TAC TOE: TIP #1

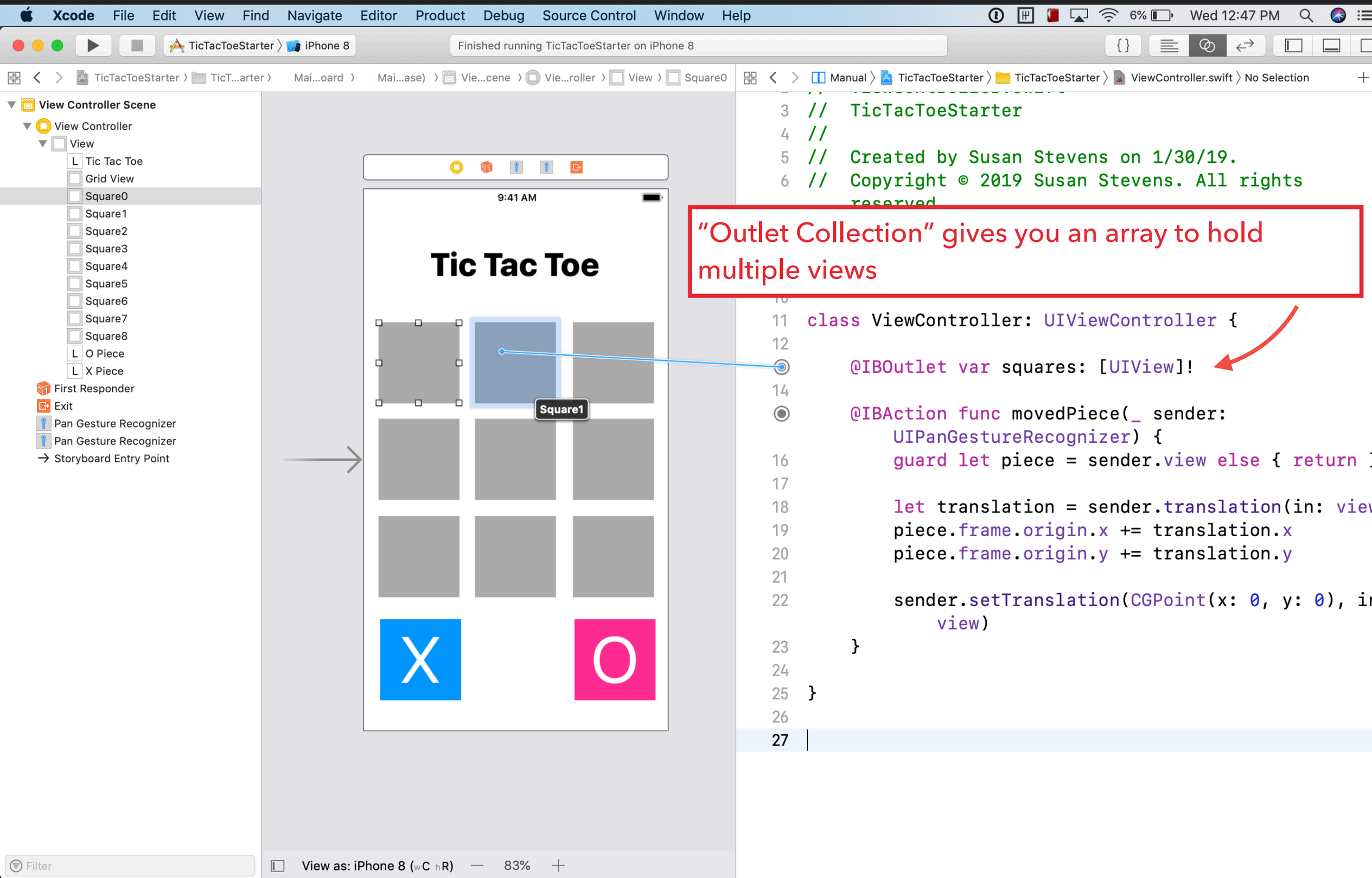
- You will need to determine when the user releases a piece over a particular square on the grid
- If you place nine transparent views on the grid, you can check if the piece overlaps with a square by calling:

```
piece.frame.intersects(square.frame)
```

Select “Outlet Collection” to create one outlet that will hold all nine squares

```
//  TicTacToeStarter
//
//  Created by Susan Stevens on 1/30/19.

import UIKit

class ViewController: UIViewController {

    @IBAction func movedPiece(_ sender:
        UIPanGestureRecognizer) {
        guard let piece = sender.view else { return }

        let translation = sender.translation(in: view)
        piece.frame.origin.x += translation.x
        piece.frame.origin.y += translation.y

        sender.setTranslation(CGPoint(x: 0, y: 0), in:
            view)
    }

}
```

ASSIGNMENT #4



TIC TAC TOE: TIP #2

- There will be a lot going on in your view controller (handling gestures and animations)
- It's a good idea to isolate logic for determining the current state of the grid into its own class

```
class ViewController: UIViewController {

    let grid = Grid()

    ...
```

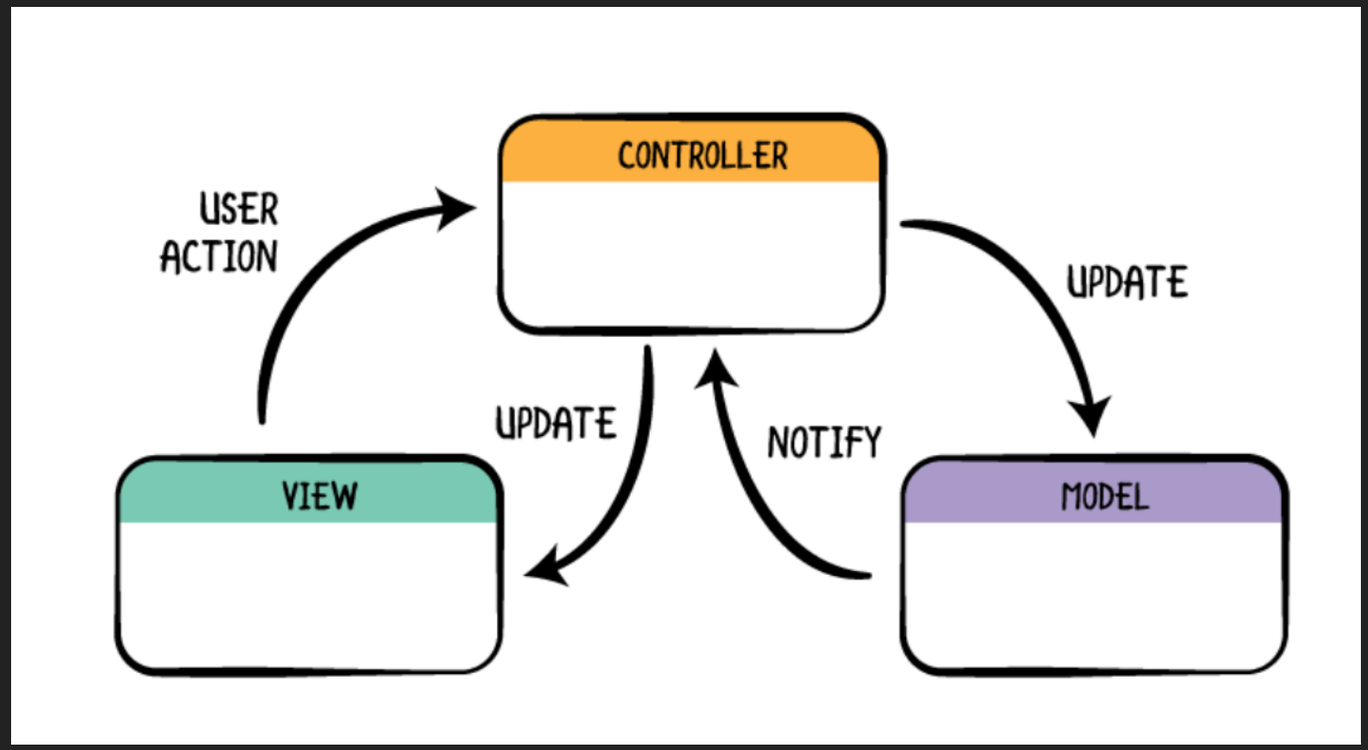
TIC TAC TOE: TIP #2

- What information about the state of the game will the view controller need from `grid`?
- What information about the view will the view controller need to provide to `grid`?

MVC

- MVC = Model-View-Controller
- The #1 design pattern in iOS

REMEMBER THIS?



https://www.raywenderlich.com/1073-model-view-controller-mvc-in-ios-a-modern-approach

MVC

- Model classes we have seen so far:
  - HW 2: `Animal`
  - HW 3: `GitHubIssue`
  - HW 4: `Grid`

TIC TAC TOE

- Due Wednesday, February 5 at 5:29pm
- Post any questions to Slack